Enumeration for FO Queries over Nowhere Dense Graphs

Nicole Schweikardt*
Humboldt-Univ. Berlin

Luc Segoufin
INRIA and ENS Ulm

Alexandre Vigny
Univ. Paris Diderot

ABSTRACT

We consider the evaluation of first-order queries over classes of databases that are *nowhere dense*. The notion of nowhere dense classes was introduced by Nešetřil and Ossona de Mendez as a formalization of classes of "sparse" graphs and generalizes many well-known classes of graphs, such as classes of bounded degree, bounded tree-width, or bounded expansion. It has recently been shown by Grohe, Kreutzer, and Siebertz that over nowhere dense classes of databases, 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 $O(n^{1+\epsilon})$, where n is the size of the database). For first-order queries of higher arities, we show that over any nowhere dense class of databases, the set of their solutions can be enumerated with constant delay after a pseudo-linear time preprocessing. In the same context, we also show that after a pseudo-linear time preprocessing we can, on input of a tuple, test in constant time whether it is a solution to the query.

ACM Reference Format:
Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. 2018. Enumeration for FO Queries over Nowhere Dense Graphs. In *PODS'18: 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3196959.3196971

1 INTRODUCTION

Query evaluation is one of the most central tasks of a database system, and a vast amount of 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 much bigger than the database itself, as the number of solutions may be exponential in the arity of the query. It can therefore be insufficient to measure the complexity of answering q on $\mathbf{D}$ only in terms of the total time needed to compute the complete result set $q(\mathbf{D})$. One can 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 on $\mathbf{D}$. 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 over all databases (this is known modulo an assumption in parameterized complexity theory, since the evaluation of boolean FO queries is AW[∗]-complete [10]). But for restricted classes of queries and databases, several efficient enumeration algorithms have been obtained. This is the case for instance for free-connex acyclic conjunctive queries over arbitrary databases [5], first-order (FO) queries over classes of databases of bounded degree [11, 19], monadic second-order (MSO) queries over classes of databases of bounded tree-width [4, 21], and FO queries over classes of databases of bounded expansion [20].

In some scenarios only pseudo-linear preprocessing time has been achieved. A query can be enumerated with constant delay after pseudo-linear preprocessing if for all ϵ there exists an enumeration procedure with constant delay (the constant may depend on ϵ) and preprocessing time in $O(\|\mathbf{D}\|^{1+\epsilon})$, where $\|\mathbf{D}\|$ denotes the size of the database. This has been achieved for FO queries over classes of databases of low degree [12] or of local bounded expansion [29].

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 can be evaluated in pseudo-linear time over nowhere dense classes of databases [16]. The notion of nowhere dense classes was introduced in [24] as a formalization of classes of "sparse" graphs and generalizes all the classes mentioned above [25]. Among classes of databases that are closed under subdatabases, the nowhere dense classes are the largest possible classes enjoying efficient evaluation of FO queries [22] (modulo an assumption in parameterized complexity theory). It has also been shown that over nowhere dense classes of databases, counting the number of solutions to a given FO query can be achieved in pseudo-linear time [17].

Main result. In this paper we show that enumeration of FO queries on nowhere dense classes of databases can be done with constant delay after pseudo-linear preprocessing. This completes the picture of the complexity of FO query evaluation on nowhere dense classes and, due to the above mentioned result of [22], on all classes that are closed under subdatabases. We also show that for

*Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SCHW 837/5-1.

PODS'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4706-8/18/06…$15.00
https://doi.org/10.1145/3196959.3196971

any nowhere dense class of databases, given a FO query q and a database $\mathbf{D}$ in the class, after a pseudo-linear time preprocessing we can test in constant time whether an arbitrary input tuple belongs to the result set $q(\mathbf{D})$.

Proof method. Our algorithms for enumerating and testing are based on the following ingredients. Instead of Gaifman's normal form (which usually serves as a starting point for algorithmic meta-theorems) we use a normal form provided by [17]. This normal form works efficiently only in the nowhere dense case and requires using explicit distance predicates in the formulas. However, it has the advantage of controlling the quantifier-rank of the local formulas. Towards evaluating local formulas we use the result that one can compute in pseudo-linear time a representative "cover" of the database by means of neighborhoods [16]. We also make use of the game characterization of nowhere dense classes [16] showing that any neighborhood can be decomposed in finitely many steps. Then, a local formula is evaluated within a neighborhood of the cover by induction on the number of remaining steps in the game until the neighborhood is trivial. The enumeration for a combination of local formulas is then done following a scheme already present in [29]: Enumeration requires precomputing pointers that allow to jump from one solution to the next one. We use a pointer mechanism similar to the one used in [29] for the local bounded expansion case.

Technical challenges. Constant delay enumeration after pseudo-linear preprocessing was already achieved for FO queries over classes of databases having bounded expansion [20] or local bounded expansion [29]. The nowhere dense case is significantly harder. The bounded expansion case was solved using a quantifier elimination procedure reducing all FO queries to the quantifier-free ones. It seems unlikely that such a quantifier elimination procedure exists in the nowhere dense case. Therefore, as for databases with local bounded expansion, the nowhere dense case relies on locality arguments and the neighborhood covers needed for solving the boolean case [16]. This was enough for databases with local bounded expansion, as the neighborhoods had bounded expansion and FO queries could then be evaluated on them. For the nowhere dense case we need a significantly more complicated argument using further tools: the game characterization of [16] and the Rank-Preserving Normal Form Theorem of [17].

Organization. The rest of the paper is structured as follows. Section 2 provides basic notations, Section 3 gives an overview of our proof, Section 4 presents our main algorithm, and Section 5 concludes the paper. Some additional examples can be found in the appendix.

2 PRELIMINARIES AND MAIN RESULT

By $\mathbb{N}$ we denote the non-negative integers, and we let $\mathbb{N}_{\geqslant 1} := \mathbb{N}\backslash\{0\}$. By $\mathbb{R}_{>0}$ we denote the set of positive reals. For $m, n \in \mathbb{N}$ we let $[m, n] := \{i \in \mathbb{N} \ : \ m \leqslant i \leqslant n\}$, and we let $[m] := [1, m]$.

Throughout this paper, ϵ will always be a positive real number, and ℓ, r, s, i, j, k will be elements of $\mathbb{N}$. For a tuple $\overline{x}$ of arity k, we will write x_i to denote its i-th component (for $i \in [k]$).

Structures and first-order queries. A relational schema is a finite set of relation symbols, each having an associated arity. A finite relational *structure* $\mathcal{A}$ over a relational schema consists of a finite set, the *domain* of $\mathcal{A}$, together with an interpretation of each relation symbol R of arity k of the schema as a k-ary relation over the domain, denoted $R(\mathcal{A})$. A *database* is a finite relational structure.

A structure $\mathcal{B}$ is a *substructure* of $\mathcal{A}$ if the domain of $\mathcal{B}$ is included in the domain of $\mathcal{A}$ and each relation of $\mathcal{B}$ is included in the corresponding relation of $\mathcal{A}$. We say that a class C of structures (or databases) is closed under substructures (or subdatabases) if for every structure $\mathcal{A}$ in C and every substructure $\mathcal{B}$ of $\mathcal{A}$ we have that $\mathcal{B}$ is in C.

If $\mathcal{A}$ is a structure with domain A and $B \subseteq A$ is a subset of its domain, we denote by $\mathcal{A}[B]$ the substructure of $\mathcal{A}$ induced by B, i.e., $\mathcal{A}[B]$ is the structure $\mathcal{B}$ with domain B and $R(\mathcal{B}) = R(\mathcal{A}) \cap B^k$ for each relation symbol R of arity k.

We fix a standard encoding of structures as input, see for example [1]. We denote by $\|\mathcal{A}\|$ the size of (the encoding of) $\mathcal{A}$, while $|\mathcal{A}|$ denotes the size $|A|$ of its domain. Without loss of generality we assume that the domain A comes with a linear order. If not, we arbitrarily choose one, for instance the one induced by the encoding of $\mathcal{A}$. This order induces a lexicographical order among the tuples over A.

A query is a first-order formula. We assume familiarity with first-order logic, FO, over relational structures (cf., e.g., [1, 23]). We use standard syntax and semantics for FO. In particular we write $q(\overline{x})$ to denote the fact that the free variables of the query q are exactly the variables in $\overline{x}$. The length of $\overline{x}$ is called the *arity* of the query. The *size* of a query q is the number of symbols needed to write down the formula and is denoted by $|q|$.

We write $\mathcal{A} \models q(\overline{a})$ to indicate that $\overline{a}$ is a solution for q over $\mathcal{A}$. We write $q(\mathcal{A})$ to denote the set of tuples $\overline{a}$ such that $\mathcal{A} \models q(\overline{a})$.

A *sentence* is a formula with no free variable, i.e. of arity 0. It is either true or false over a structure and therefore defines a property of structures, i.e., a boolean query. Given a relational structure $\mathcal{A}$ and a sentence q, the problem of testing whether $\mathcal{A} \models q$ is called *the model checking problem*. Often, the problem is restricted to a particular class C of relational structures.

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

All problems encountered in this paper have two inputs: a structure $\mathcal{A}$ and a query $q(\overline{x})$. However, they play different roles as $\|\mathcal{A}\|$ is often very large while $|q|$ is generally small. We adopt the data complexity point of view [31]. When we say *linear time* we mean in time $O(\|\mathcal{A}\|)$, the constants hidden behind the "big O" depending on q, on the class C of structures under investigation, and possibly on further parameters that will be clear from the context. We say that a problem is solvable in *pseudo-linear time* if, for all ϵ, it can be solved in time $O(\|\mathcal{A}\|^{1+\epsilon})$. In this case, the constant factor also depends on ϵ. If a subroutine of a procedure depending on ϵ produces an output of size $O(\|\mathcal{A}\|^{\epsilon})$ we will say that the output is *pseudo-constant*.

Distance and neighborhoods. Fix a structure $\mathcal{A}$ of domain A. The *Gaifman graph* of $\mathcal{A}$ is the undirected graph whose set of

vertices is A and whose edges are the pairs $\{a, b\}$ such that a and b occur in a tuple of some relation of $\mathcal{A}$. Given two elements a and b of A, the *distance* between a and b is the length of a shortest path between a and b in the Gaifman graph of $\mathcal{A}$. The notion of distance extends to tuples in the usual way, i.e., the distance between two tuples $\overline{a}$ and $\overline{b}$ is the minimum of the distances between a_i and b_j over all i, j.

For a positive integer r, we write $N_r^{\mathcal{A}}(a)$ for the set of all elements of A at distance at most r from a. The r-neighborhood of a in $\mathcal{A}$, denoted $\mathcal{N}_r^{\mathcal{A}}(a)$, is the substructure of $\mathcal{A}$ induced by $N_r^{\mathcal{A}}(a)$. Similarly, for a tuple $\overline{a}$ of arity k we let $N_r^{\mathcal{A}}(\overline{a}) := \bigcup_{i\in[k]} N_r^{\mathcal{A}}(a_i)$, and we define $\mathcal{N}_r^{\mathcal{A}}(\overline{a})$ as the substructure of $\mathcal{A}$ induced by $N_r^{\mathcal{A}}(\overline{a})$.

FO$^+$ and q-rank. As the classical version of Gaifman's locality theorem [15] is not powerful enough for our needs, we will use another local normal form, recently introduced in [17]. This normal form is computed for a syntactic extension of first-order logic called FO$^+$. In FO$^+$ we are allowed to use atoms of the form $\mathrm{dist}(x, y) \leqslant d$ for any constant d, and when evaluated in a structure $\mathcal{A}$, $\mathrm{dist}(x, y)$ is interpreted as the distance between x and y in the Gaifman graph of $\mathcal{A}$.

Allowing to use these distance-atoms does not increase the expressive power of first-order logic, as these atoms can clearly be expressed in FO, but it will lead to a different notion of quantifier-rank. Following [16], we say that a FO$^+$ query φ has *q-rank at most ℓ*, where q and ℓ are in $\mathbb{N}$, if it has quantifier-rank at most ℓ and each distance-atom $\mathrm{dist}(x, y) \leqslant d$ in the scope of $i \leqslant \ell$ quantifiers satisfies $d \leqslant (4q)^{q+\ell-i}$. We also define (as in [16] and [17])

$$f_q(\ell) \quad := \quad (4q)^{q+\ell}.$$

Nowhere dense classes of undirected graphs. For an undirected graph $G = (V, E)$ we let $|G| = |V|$ and $\|G\| = |V| + |E|$. Thus, similarly as for databases, $|G|$ denotes the size of the graph's domain, and $\|G\|$ is the size of a reasonable encoding of G.

Given an undirected 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 a and all other vertices on the path. A class C of undirected graphs is *nowhere dense* if for all r and ϵ, there is a number N, such that for all graphs G of C with $|G| > N$, 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 $|G|^\epsilon$ [25]. This is a robust notion, and there exist many equivalent definitions [16, 24, 25]; in Section 3 we will use the game characterization of [16]. It is known, for example, that any class of graphs that (locally) excludes a minor or that is of (local) bounded expansion, is nowhere dense.

Nowhere dense classes are "sparse" in the following sense: For every nowhere dense class C of undirected graphs there is a function f such that for every $\epsilon > 0$ and every G in C, if $|G| \geqslant f(\epsilon)$, then $\|G\| \leqslant |G|^{1+\epsilon}$ (see [26]).

In the special case where for all integers r there is a constant c_r such that for all graphs G of C, and all vertices a of G the number of vertices weakly r-accessible from a is bounded by c_r, the class C is said to have *bounded expansion*. These constants were the keys to the constant delay enumeration algorithm for FO queries over classes of bounded expansion [20]. As we no longer have them in the nowhere dense case, we need a different strategy.

From databases to colored graphs. We define *c-colored graphs* as structures over the schema $\sigma_c := \{E, C_1, \ldots, C_c\}$ where E is a binary relation and $(C_i)_{i\leqslant c}$ are unary relations. A *colored graph* is a c-colored graph for some integer c.

A class C of colored graphs is defined to be *nowhere dense* if the class C' consisting of the underlying undirected graphs of all elements of C is nowhere dense.

Our main enumeration algorithm works for FO-queries on nowhere dense classes of colored graphs. By using standard techniques, this extends to all relational structures, as we now explain.

Given a database $\mathbf{D}$ over a schema σ, we define its *adjacency graph* $A(\mathbf{D})$ as the relational structure whose domain is $D \cup T$ where D is the domain of $\mathbf{D}$ and T is the set of tuples occurring in a relation of $\mathbf{D}$. We have one unary relation P_R per relation R of σ containing all tuples t of $R(\mathbf{D})$. We have k binary relations $E_1, \ldots, E_k$ where k is the maximal arity of relations in σ. For $a \in D$ and $t \in T$, we have $A(\mathbf{D}) \models E_i(a, t)$ if and only if the element a is the i^{th} element of the tuple t.

The *colored graph version* $A'(\mathbf{D})$ of the adjacency graph $A(\mathbf{D})$ is defined as follows.

The colors of $A'(\mathbf{D})$ are the "vertex colors" P_R of $A(\mathbf{D})$ and k further colors $(C_i)_{i\leqslant k}$, where k is the maximal arity of the relations in σ. The domain of $A'(\mathbf{D})$ is the domain of $A(\mathbf{D})$ plus one node per edge of $A(\mathbf{D})$: For every E_i-edge (a, t) of $A(\mathbf{D})$, we add in $A'(\mathbf{D})$ a new node v of color C_i and such that (a, v) and (v, t) are E-edges in $A'(\mathbf{D})$. In particular, $A'(\mathbf{D})$ is a colored graph of schema $\{E, C_1, \ldots, C_k, P_{R_1}, \ldots, P_{R_m}\}$ if $\mathbf{D}$ is a database of schema $\sigma = \{R_1, \ldots, R_m\}$ consisting of m relations of maximum arity k. Henceforth, we will identify the schema of $A'(\mathbf{D})$ with the schema σ_c of c-colored graphs for $c = k + m$.

The following straightforward lemma reduces relational structures to colored graphs.

Lemma 2.1. *Let $\varphi(\overline{x})$ be a* FO *query over some schema σ and let k be the maximal arity of the relations of σ. In time linear in the size of φ, we can compute a* FO *query $\psi(\overline{x})$ over σ_c, for $c = k + |\sigma|$, such that for every database $\mathbf{D}$ over σ, $\varphi(\mathbf{D}) = \psi(A'(\mathbf{D}))$.*

The lemma is a consequence of the fact that

$$\mathbf{D} \;\models\; R(a_1, \ldots, a_j) \quad \Longleftrightarrow$$

$$A'(\mathbf{D}) \;\models\; \exists t\Big(P_R(t) \wedge \bigwedge_{i\leqslant j} \exists z\big(C_i(z) \wedge E(a_i, z) \wedge E(z, t)\big)\Big).$$

Let C be a class of relational structures. We say that C is *nowhere dense* if the class $\{A'(\mathbf{D}) \;:\; \mathbf{D} \in C\}$ is nowhere dense. Note that because $A'(\mathbf{D})$ is a 1-subdivision of $A(\mathbf{D})$, the class $\{A'(\mathbf{D}) \;:\; \mathbf{D} \in C\}$ is nowhere dense iff the class consisting of the Gaifman graphs of $A(\mathbf{D})$ for all $\mathbf{D} \in C$ is nowhere dense; see [25] for further details.

We could have used another definition for nowhere dense classes of structures, using their Gaifman graphs instead of their adjacency graphs. For a fixed schema this would result in the same notion, but when the schema is not fixed our definition is more general [18].

A consequence of Lemma 2.1 is that the model checking, enumeration, counting, and testing problems for FO-queries reduce to the colored graph case. In the remaining part of the paper we

will therefore only consider classes of colored graphs. The reader should keep in mind, though, that the results stated over colored graphs extend to relational structures.

Enumeration. An *enumeration* algorithm for a colored graph G and a query q is divided into two consecutive phases:

- a preprocessing phase, and
- an enumeration phase, outputting one by one and without repetition the elements of the set $q(G)$.

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 that computes a representation of $q(G)$, together with a streaming decompression algorithm.

We aim for enumeration algorithms with constant delay and pseudo-linear preprocessing time. By this we mean that for all ϵ, there is a preprocessing phase working in time $O(\|G\|^{1+\epsilon})$ 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 ϵ.

An enumeration phase with constant delay may use a constant amount of memory while preparing the next output. Hence it may use a total amount of memory that is linear in the output size. Our enumeration algorithms will have the property that, apart from the memory used for storing the data structure (of pseudo-linear size) that is produced during the preprocessing phase, the entire enumeration phase only uses a constant amount of extra memory. In other words, our enumeration algorithm can be seen as a finite state automaton running on the index structure produced by the preprocessing phase.

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

Our main result. We now state our main theorem.

Theorem 2.2. *Let $q(\overline{x})$ be a first-order query and $\mathcal{C}$ a nowhere dense class of colored graphs. There is an algorithm which, on input of a colored graph $G \in \mathcal{C}$, enumerates $q(G)$ lexicographically, with constant delay after a pseudo-linear preprocessing phase. Moreover, after this preprocessing, if we are given a tuple $\overline{a}$, we can decide in constant time whether $\overline{a} \in q(G)$.*

3 PROOF SKETCH

When given a c-colored graph G, we will always write V for its domain, i.e. the set of its nodes, and E for the set of its edges. For a $v \in V$ we will often write $G \setminus \{v\}$ instead of $G[V \setminus \{v\}]$. For $c' > c$, a *$\sigma_{c'}$-expansion* of a c-colored graph G is a c'-colored graph G' which coincides with G in its domain, its edge set, and its first c unary relations $C_1, \ldots, C_c$.

Throughout the remainder of this paper, we will briefly write "graphs" instead of "colored graphs".

In this section we describe the proof idea and introduce notions that are needed for the proof of Theorem 2.2. They are illustrated using a running example that forms a sketch of the proof.

The proof of Theorem 2.2 proceeds by induction on the number of free variables. Theorem 3.1 below takes care of the base cases, arity 0 and 1.

Theorem 3.1 (Grohe, Kreutzer, Siebertz [16])**.** *Let q be a first-order query with at most one free variable and $\mathcal{C}$ a nowhere dense class of graphs. There is an algorithm that, on input of $G \in \mathcal{C}$ computes $q(G)$ in pseudo-linear time.*

Example (1-A)**.** *Consider the distance 2 query:*

$$q(x,y) \quad := \quad \exists z \left(E(x,z) \wedge E(z,y) \right)$$

To enumerate all pairs (a,b) such that $G \models q(a,b)$, we first use Theorem 3.1 to compute the list of all a such that $G \models \exists y\, q(a,y)$. It remains to construct an index structure such that, on input an element a in this list, we can enumerate with constant delay all b such that $G \models q(a,b)$.

It is well-known that first-order queries are local in the sense that whether or not a tuple belongs to the query result $q(G)$ only depends on the r-neighborhood of the tuple, where the radius r depends only on the query, but not on G. It is therefore tempting to precompute the r-neighborhoods of all nodes. Unfortunately, this cannot be done in pseudo-linear time as the sum of the sizes of those neighborhoods might be too big. To overcome this situation we use a *neighborhood cover* that selects a small but sufficiently representative set of neighborhoods [2, 3, 28] and that has been utilized for FO query evaluation already in [13, 16, 29].

Definition 3.2 (Neighborhood cover)**.** Given a graph G and a number $r \in \mathbb{N}$, an *r-neighborhood cover* of G is a collection $\mathcal{X}$ of subsets of V such that

$$\forall\, a \in V \;\; \exists X \in \mathcal{X} \;\text{ with }\; N_r^G(a) \subseteq X.$$

For $r, s \in \mathbb{N}$, an *(r,s)-neighborhood cover* of G is an r-neighborhood cover of G satisfying

$$\forall\, X \in \mathcal{X} \;\; \exists\, a \in V \;\text{ with }\; X \subseteq N_s^G(a).$$

The number r is called the *radius* of the neighborhood cover, and the sets $X \in \mathcal{X}$ are called *bags*.

The *degree* $\delta(\mathcal{X})$ of the cover is the maximal number of bags that intersect at a given node, i.e.

$$\delta(\mathcal{X}) \quad := \quad \max_{a \in V} |\{X \in \mathcal{X} \;:\; a \in X\}|.$$

Given a neighborhood cover of radius r of G containing m bags, we fix an arbitrary order among the bags and denote them with $X_1, \ldots, X_m$. In the sequel, by *computing a neighborhood cover* of radius r with m bags, we mean the following: To every integer $i \leqslant m$ we associate the content of the i^{th} bag X_i of the cover so that, given i we can later access in constant time to X_i. Additionally, to every element a we choose arbitrarily a bag X_i containing its r-neighborhood and compute a function associating i to a. The bag X_i is later denoted $\mathcal{X}(a)$.

Neighborhood covers with small degree can be computed efficiently on nowhere dense classes of graphs:

Theorem 3.3 ([16])**.** *Let $\mathcal{C}$ be a nowhere dense class of graphs. Then there are a function $f_{\mathcal{C}} : \mathbb{N} \times \mathbb{R}_{>0} \to \mathbb{N}$ and an algorithm that, given an $\epsilon > 0$, an $r \in \mathbb{N}$, and a $G \in \mathcal{C}$ whose domain V is larger than $f_{\mathcal{C}}(r,\epsilon)$, computes in time $f_{\mathcal{C}}(r,\epsilon) \cdot |V|^{1+\epsilon}$ an $(r,2r)$-neighborhood cover of G with degree at most $|V|^{\epsilon}$.*

We will also make use of the following notion of kernel that has been utilized for FO query evaluation already in [13, 16, 29].

Definition 3.4 (Kernel). If $\mathcal{X}$ is a neighborhood cover of G with radius r, we define for all $X \in \mathcal{X}$ and $p \leqslant r$, the *p-kernel* of X as the set $K_p(X) := \{a \in V \ : \ N_p^G(a) \subseteq X\}$.

Lemma 3.5 ([16]). *Assume $\mathcal{X}$ is a neighborhood cover of G. Given a bag X and a number p, we can compute $K_p(X)$ in time $O(p \cdot \|G[X]\|)$.*

Let's get back to our running example.

Example (1-B). *The radius of the relevant cover depends on the query. For our query q from Example (1-A), we have for all nodes a, b that*

$$G \models q(a,b) \quad \iff \quad \mathcal{N}_2^G(a) \models q(a,b).$$

Therefore if we compute a $(2,4)$-neighborhood cover $\mathcal{X}$ of G, since $N_2^G(a) \subseteq \mathcal{X}(a)$, we have that for all nodes a and b:

$$G \models q(a,b) \quad \iff \quad G[\mathcal{X}(a)] \models q(a,b)$$

and

$$G[\mathcal{X}(a)] \models q(a,b) \quad \iff \quad \bigvee_{X \in \mathcal{X}} G[X] \models q(a,b).$$

Hence, modulo a pseudo-linear preprocessing, given an element a, to enumerate all b at distance 2 from a, it is enough to restrain our attention to the bag $\mathcal{X}(a)$ of a. We will see later how this will be useful.

As illustrated by our running example, we will reduce the problem to enumerating the query results inside a bag of the cover. The following game characterization of nowhere dense classes of graphs will give us a second inductive parameter that will decrease when diving within a bag (recall that our first inductive parameter is the number of free variables).

Definition 3.6 (Splitter game [16]). Let G be a graph and let $\lambda, r \in \mathbb{N}_{\geqslant 1}$. The *$(\lambda, r)$-splitter game* on G is played by two players called *Connector* and *Splitter*, as follows. We let $G_0 := G$ and $V_0 := V$. In round $i+1$ of the game, Connector chooses a vertex $a_{i+1} \in V_i$. Then Splitter picks a vertex $b_{i+1} \in N_r^{G_i}(a_{i+1})$. If $V_{i+1} := N_r^{G_i}(a_{i+1}) \setminus \{b_{i+1}\} = \emptyset$, then Splitter wins the game. Otherwise, the game continues with $G_{i+1} := G_i[V_{i+1}]$. If Splitter has not won after λ rounds, then Connector wins.

This is not exactly the same definition as the one in [16], where several nodes can be removed in one step, but it is easy to show that it is equivalent to it [30]. We say that Splitter *wins* the (λ, r)-splitter game on G if she has a winning strategy for the game.

Theorem 3.7 ([16]). *A class C of graphs is nowhere dense if, and only if, for every $r \in \mathbb{N}_{\geqslant 1}$ there is a $\lambda(r) \in \mathbb{N}_{\geqslant 1}$, such that for every $G \in C$, Splitter wins the $(\lambda(r), r)$-splitter game on G.*

Our second inductive parameter will be the number of remaining rounds for Splitter before she wins the game starting with parameters $(\lambda(r'), r')$ when given an $(r'/2, r')$-neighborhood cover of G for a suitable number r'.

Example (1-C). *For our running example, we have already computed a $(2,4)$-neighborhood cover $\mathcal{X}$ of G. Let then λ be such that Splitter wins the $(\lambda, 4)$-splitter game on G. Assume that $\lambda = 1$. Then G is edgeless and any naive algorithm will work. Assume now that $\lambda > 1$. Let X be a bag of $\mathcal{X}$. By definition, there is an element u such that $X \subseteq N_4^G(u)$. Let v be Splitter's answer if Connector picks u in the game's first round. Then, by definition, Splitter wins the $(\lambda-1, 4)$-splitter game on $G[N_4^G(u) \setminus \{v\}]$. In particular, Splitter wins the $(\lambda-1, 4)$-splitter game on $G' := G[X \setminus \{v\}]$. Hence we can use an inductive argument within G'. For this we compute a new query $q'(x,y)$ such that for all a, b such that $\mathcal{X}(a) = X$, $G \models q(a,b)$ iff $G' \models q'(a,b)$. Recall that we already know for such pairs a, b that $G \models q(a,b)$ iff $G[X] \models q(a,b)$. It therefore remains to encode the removal of the node v, and this can be done by recoloring the nodes adjacent to v. Let $R_1, \cdots, R_4$ be new (and materialized) unary predicates such that:*

$$\begin{array}{lll} w \in R_1(G') & \text{iff} & G \models E(w,v) \\ w \in R_2(G') & \text{iff} & G \models E(v,w) \\ w \in R_3(G') & \text{iff} & G \models q(w,v) \\ w \in R_4(G') & \text{iff} & G \models q(v,w). \end{array}$$

The new query $q'(x,y)$ is then defined as a disjunction of the following queries

$$\begin{array}{rcl} q(x,y) & \vee & \Big(R_1(x) \ \wedge \ R_2(y) \Big) \\ & \vee & \Big(R_3(x) \ \wedge \ y = v \Big) \\ & \vee & \Big(R_4(y) \ \wedge \ x = v \Big). \end{array}$$

The first disjunct takes care of the general case, the second one for when the unique node connecting x and y is v, the third case when $y = v$ and the last one. As v is not part of G', the case $R_3(x) \ \wedge \ y = v$ should be understood as listing all x in G' such that $G' \models R_3(x)$ and then outputting the pair (x, v). Similarly for the last case.

The following lemma generalizes the idea of the example by rewriting a query into an equivalent one when a node is removed. The lemma is present in a similar form in [17]; its proof is straightforward. For a tuple $\overline{a}$ and a set I of indices, we denote by $\overline{a}_I$ the projection of $\overline{a}$ onto its components whose indices belong to I. By $\overline{a}_{\setminus I}$ we denote the projection of $\overline{a}$ onto its components whose indices are *not* in I. Recall from Section 2 the notion of FO^+-formulas of q-rank at most ℓ.

Lemma 3.8 (Removal Lemma). *Let $k, q, \ell \in \mathbb{N}$. There is an algorithm which takes as input a number $c \in \mathbb{N}$, a c-colored graph G, a k-ary query $\varphi(\overline{z}) \in \mathrm{FO}^+[\sigma_c]$ of q-rank at most ℓ, a set of free variables $\overline{y} \subseteq \overline{z}$, and a node $v \in V$, and which produces*

- *a number $c' \geqslant c$,*
- *a query $\varphi'(\overline{z} \setminus \overline{y}) \in \mathrm{FO}^+[\sigma_{c'}]$ of q-rank at most ℓ,*
- *a $\sigma_{c'}$-expansion H of $G \setminus \{v\}$,*

such that for all tuples $\overline{b}$ over V where $I := \{i \leqslant k \ : \ b_i = v\} = \{i \leqslant k \ : \ z_i \in \overline{y}\}$ we have

$$G \models \varphi(\overline{b}) \quad \iff \quad H \models \varphi'(\overline{b}_{\setminus I}).$$

Moreover, the running time of this algorithm is linear in the size of G, and

- *c' only depends on c, q, ℓ,*
- *φ' only depends on $c, q, \ell, \varphi, \overline{y}$,*
- *H only depends on c, q, ℓ, G, v.*

For our proof it is important that the radius of the neighborhood cover, and therefore the number of rounds Splitter needs in the game, are fixed once and for all at the beginning of the induction. It is well-known that this radius depends on the quantifier-rank of the formula. Unfortunately, the classical Gaifman Normal Form Theorem [15] does not control the quantifier-rank of the local

formulas it generates. As we would like to use it inductively for the queries derived by Lemma 3.8, the radius may change during the induction — and we don't want this to happen. As a remedy, we use a locality theorem based on q-rank instead of quantifier-rank. It has the advantage that it preserves the q-rank within the local formulas. It was first introduced in [16] for unary queries. Here, we need a stronger variant for queries of arbitrary arities, proved in [17]. For formulating the theorem, we need the following notation.

An (r,q)-*independence sentence* is an FO^+-sentence of the form

$$\exists z_1 \cdots \exists z_{k'} \Big(\bigwedge_{1 \leqslant i < j \leqslant k'} \mathrm{dist}(z_i, z_j) > r' \ \wedge \ \bigwedge_{1 \leqslant i \leqslant k'} \psi(z_i) \Big)$$

where $k' \leqslant q$ and $r' \leqslant r$ and $\psi(z)$ is quantifier-free.

Given a graph G and a tuple $\overline{a} = (a_1, \ldots, a_k) \in V^k$, for all $r \in \mathbb{N}$ we define the r-*distance type* of $\overline{a}$ as the undirected graph $\tau_r^G(\overline{a})$ whose nodes are $[1,k]$, and where $\{i,j\}$ is an edge iff $G \models \mathrm{dist}(a_i, a_j) \leqslant r$. The set of all possible distance types with k elements is denoted $\mathcal{T}_k$.

For a colored graph G, a neighborhood cover $\mathcal{X}$ of G, a number $r \in \mathbb{N}$, and a tuple $\overline{a}$ of nodes of G we say that a bag $X \in \mathcal{X}$ r-*covers* $\overline{a}$ if $N_r^G(\overline{a}) \subseteq X$.

Theorem 3.9 (Rank-Preserving Normal Form [17])**.** *Let $q, k \in \mathbb{N}$ such that $k \leqslant q$, let $\ell := q - k$, $r := f_q(\ell)$. For every $c \in \mathbb{N}$ we can compute a $c' \geqslant c$ such that the following is true for $\sigma := \sigma_c$ and $\sigma^\star := \sigma_{c'}$. For every $\mathrm{FO}^+[\sigma]$-formula $\varphi(\overline{x})$ of q-rank at most ℓ where $\overline{x} = (x_1, \ldots, x_k)$, and for each distance type $\tau \in \mathcal{T}_k$ we can compute a number $m_\tau \in \mathbb{N}$ and, for each $i \leqslant m_\tau$,*

- *a Boolean combination ξ_τ^i of (r,q)-independence sentences of schema $\sigma^\star$ and*
- *for each connected component I of τ an $\mathrm{FO}^+[\sigma^\star]$-formula $\psi_{\tau,I}^i(\overline{x}_I)$ of q-rank at most ℓ*

such that the following holds:

For all c-colored graphs G and all kr-neighborhood covers $\mathcal{X}$ of G, there is a $\sigma^\star$-expansion $G^\star$ of G (which only depends on $G, \mathcal{X}, q, \ell$) with the following properties, where V denotes the domain of G and $G^\star$:

(a) For all $\overline{a} \in V^k$ and for $\tau := \tau_r^G(\overline{a})$ we have:
$G \models \varphi(\overline{a})$ iff there is an $i \leqslant m_\tau$ that satisfies the following condition
$(*)_i$: $G^\star \models \xi_\tau^i$ and for every connected component I of τ there is an $X \in \mathcal{X}$ that r-covers $\overline{a}_I$ and $G^\star[X] \models \psi_{\tau,I}^i(\overline{a}_I)$.

(b) For all $\overline{a} \in V^k$ and for $\tau := \tau_r^G(\overline{a})$, there is at most one $i \leqslant m_\tau$ such that the condition $(*)_i$ is satisfied.

(c) For all $\overline{a} \in V^k$, all connected components I of $\tau := \tau_r^G(\overline{a})$, all $X, X' \in \mathcal{X}$ that both r-cover $\overline{a}_I$, and all $i \leqslant m_\tau$,

$$G^\star[X] \models \psi_{\tau,I}^i(\overline{a}_I) \iff G^\star[X'] \models \psi_{\tau,I}^i(\overline{a}_I).$$

Furthermore, for every nowhere dense class $\mathcal{C}$ of colored graphs, there is an algorithm which, when given as input a $G \in \mathcal{C}$, a kr-neighborhood cover $\mathcal{X}$ of G of degree at most $|V|^\epsilon$, and parameters $q, \ell \in \mathbb{N}$, computes $G^\star$ in time $O(|V|^{1+2\epsilon})$.

Example (2)**.** *Our first query example is limited in the sense that all its solutions satisfy the same distance type requiring that x is close to y. We therefore consider a slightly more complicated query:*

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

where B is interpreted as the set of "blue" nodes of a colored graph. As before, the goal is, given a node a, to enumerate all blue nodes that are at distance greater than 2 from a. As previously, we compute a $(2,4)$-neighborhood cover, and given a node a, we consider the bag $X := \mathcal{X}(a)$.

We then start two concurrent enumeration processes: the first one only enumerates nodes that are within X; the second one enumerates those that are not in X. The first one uses ideas previously presented, diving into $G[X \setminus \{v\}]$ using the query constructed by Lemma 3.8 and the induction on λ. We now explain how the second one works.

Note that for every node b outside of X, we have $\mathrm{dist}(a,b) > 2$ as $N_2^G(a) \subseteq X$. Therefore, it suffices to enumerate all blue nodes that don't belong to X. To do so, during the preprocessing phase we compute for all nodes c of G and all bags $X \in \mathcal{X}$ with $c \in X$, the smallest blue node bigger than c that is not in X; let us denote this node by $v(c,X)$. As the degree of our cover is pseudo-constant, the domain of the function $v(\cdot,\cdot)$ is pseudo-linear and the computation of the function can be done in pseudo-linear time.

However, a naive extension of this idea for queries of bigger arities does not work: consider the query

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

Assume that, given a pair (a,b) of nodes, we want to enumerate all nodes c such that $q(a,b,c)$ holds. Given a, b we consider the bags $X := \mathcal{X}(a)$ and $Y := \mathcal{X}(b)$. Now, we have three concurrent processes, one of them being in charge of enumerating all blue nodes that are neither in X nor in Y. The previous algorithm can enumerate all blue nodes that are not in X, but some of those may be in Y. Computing given c in $X \cup Y$ the smallest blue node c' bigger than c which falls out of $X \cup Y$, may require quadratic time and space. Therefore, we need another approach.

The following lemma takes care of the issue raised in the previous example.

Lemma 3.10 (Skip pointers [29])**.** *For every nowhere dense class $\mathcal{C}$ of colored graphs, for every G in $\mathcal{C}$, for every $r > 0$, every $\epsilon > 0$, and every $k \in \mathbb{N}$, for every r-neighborhood cover $\mathcal{X}$ of G of degree at most $|V|^\epsilon$, and for every $L \subseteq V$, there is a preprocessing algorithm working in time $O(|V|^{1+k\epsilon})$ allowing us, when given a node b and a set S of at most k bags of $\mathcal{X}$, to compute in constant time the node*

$$\mathrm{SKIP}(b,S) \quad := \quad \min \Big\{ b' \in L \,:\, b' \geqslant b \ \wedge \ b' \notin \bigcup_{X \in S} K_r(X) \Big\}$$

(recall from Definition 3.4 that $K_r(X)$ is the r-kernel of X).

Proof. The lemma was proved already in [29], but in order to make the current paper a bit more self-contained let us recapitulate the proof details here.

From now on we fix ϵ, r, G, $\mathcal{X}$ and L as in the statement of the lemma.

We assume that all kernels have already been computed. This is without loss of generality modulo a preprocessing of time $O(\|G\|^{1+\epsilon})$ using Lemma 3.5.

The domain of the $\mathrm{SKIP}(\cdot,\cdot)$-function is too big (recall that there can be a linear number of bags) so we cannot compute it during the preprocessing phase. Fortunately, computing only a small part of it will be good enough for our needs. For each node b we define

by induction a set $SC(b)$ of sets of at most k bags. We start with $SC(b) = \emptyset$ and then proceed as follows.

- For all nodes b of G and for all bags X in $\mathcal{X}$ with $b \in K_r(X)$, we add $\{X\}$ to $SC(b)$.
- For all nodes b of G, for all sets S of bags from $\mathcal{X}$, and all bags X of $\mathcal{X}$, if $|S| < k$ and $S \in SC(b)$ and $\mathrm{SKIP}(b, S) \in K_r(X)$, then we add $\{S \cup \{X\}\}$ to $SC(b)$.

In the preprocessing phase we will compute $\mathrm{SKIP}(b, S)$ for all nodes b of G and all sets $S \in SC(b)$. Before explaining how this can be accomplished within the desired time constraints, we first show that this is sufficient for deriving $\mathrm{SKIP}(b, S)$ in constant time for all nodes b and all sets S consisting of at most k bags of $\mathcal{X}$.

Claim 3.11. *Given a node b of G, a set S of at most k bags of $\mathcal{X}$, and $\mathrm{SKIP}(c, S')$ for all nodes $c > b$ of G and all sets $S' \in SC(c)$, we can compute $\mathrm{SKIP}(b, S)$ in constant time.*

Proof. We consider two cases (testing in which case we fall can be done in constant time as the kernels have been computed and S has size bounded by k).

Case 1: $b \in L$ and $b \notin \bigcup_{X \in S} K_r(X)$. In this case, b is $\mathrm{SKIP}(b, S)$ and we are done.

Case 2: $b \notin L$ or $b \in \bigcup_{X \in S} K_r(X)$. In this case, let c be the smallest element of L strictly bigger than b. If there is no such c then $\mathrm{SKIP}(b, S) = \mathsf{Null}$ and we are done. Otherwise, we proceed as follows.

If $c \notin \bigcup_{X \in S} K_r(X)$, then c is $\mathrm{SKIP}(b, S)$ and we are done. Otherwise, we know that $c \in K_r(X)$ for some $X \in S$. Therefore $\{X\} \in SC(c)$. Let S' be a maximal (w.r.t. inclusion) subset of S in $SC(c)$. Since $\{X\} \in SC(c)$, we know that S' is non-empty.

We claim that $\mathrm{SKIP}(c, S') = \mathrm{SKIP}(b, S)$. To prove this, let us first assume for contradiction that $\mathrm{SKIP}(c, S') \in K_r(Y)$ for some $Y \in S$. By definition, this implies that Y is not in S'. Hence $|S'| < |S| \leqslant k$. Thus, by definition of $SC(c)$ we have $S' \cup \{Y\} \in SC(c)$ and S' was not maximal.

Moreover, by definition of $\mathrm{SKIP}(c, S')$, every point between c and $\mathrm{SKIP}(c, S')$ is either not in L or in some $K_r(Z)$ with $Z \in S'$ (and therefore $Z \in S$). As all nodes between b and c are not in L, the claim follows. □

We conclude by showing that $SC(b)$ is small for all nodes b of G and that we can compute efficiently $\mathrm{SKIP}(b, S)$ for all nodes b and all sets $S \in SC(b)$.

Claim 3.12. *For each node b of G, $|SC(b)|$ has size $O(|V|^{k\epsilon})$. Moreover, it is possible to compute $\mathrm{SKIP}(b, S)$ for all nodes b of G and all sets $S \in SC(b)$ in time $O(|V|^{1+k\epsilon})$*

Proof. We start by proving the first statement, and afterwards we use Claim 3.11 to show that we can compute these pointers inductively.

By $SC_\ell(b)$ we denote the subset of $SC(b)$ of sets S with $|S| \leqslant \ell$. Let d be the degree of the cover $\mathcal{X}$, i.e., $d = |V|^\epsilon$. By definition of d, we know that $|SC_1(b)| \leqslant d$ for all nodes b of G. For the same reason, we have that $|SC_{\ell+1}(b)|$ is of size at most $O(d \cdot |SC_\ell(b)|)$. Therefore, for all $b \in V$, we have:

$$|SC(b)| \;=\; |SC_k(b)| \;\leqslant\; O(d^k).$$

We compute the pointers for b from b_{max} to b_{min} downwards, where b_{max} and b_{min} are, respectively, the biggest and the smallest element of V. Given a node b in V, assume we have computed $\mathrm{SKIP}(c, S')$ for all $c > b$ and $S' \in SC(c)$. We then compute $\mathrm{SKIP}(b, S)$ for $S \in SC(b)$ using Claim 3.11.

At each step the pointer is computed in constant time. Since there are $O(|V|^{1+k\epsilon})$ of them, the time required to compute them is as desired. □

The combination of these two claims proves Lemma 3.10. □

Finally, we have available all technical notions and results that we need for presenting our constant-delay algorithm for enumerating the result of first-order queries.

4 MAIN ALGORITHM

This section is devoted to the proof of Theorem 2.2. Let C be a fixed nowhere dense class of colored graphs. Then, also the class C' consisting of the Gaifman graphs of all elements in C is nowhere dense. W.l.o.g. we assume that C' is closed under taking subgraphs and that C is the class of all colored graphs (with aritrarily large numbers of colors) whose Gaifman graph is in C'.

We prove Theorem 2.2 by induction on the arity k of the given query φ. Theorem 3.1 takes care of the induction base for $k = 0$ and $k = 1$, as precomputing the list of solutions is enough for our needs. For the induction step let us consider some arity $k \geqslant 2$ (that is fixed throughout the remainder of this section) and assume that the statement of Theorem 2.2 has already been proven for all queries of arity $< k$.

Now let us fix an arbitrary number $q \in \mathbb{N}$ with $q \geqslant k$, let $\ell := q-k$, and let $r := f_q(\ell)$. Note that this is the same choice of parameters as for the Rank Preserving Normal Form Theorem 3.9. Our goal is to show that the statement of Theorem 2.2 is true for all k-ary queries φ of q-rank at most ℓ. We prove the following, where $\overline{x} = (x_1, \ldots, x_{k-1})$ is a $(k-1)$-ary tuple of variables and x_k is a further variable.

Proposition 4.1. *For all $\epsilon > 0$ and all FO^+-queries $\varphi(\overline{x}, x_k)$ of arity k and q-rank at most ℓ there exists an algorithm which, upon input of a colored graph $G \in C$, performs a preprocessing in time $O(|V|^{1+\epsilon})$, such that afterwards*

- *upon input of a tuple $(\overline{a}, b) \in V^k$, we can test in constant time whether $G \models \varphi(\overline{a}, b)$, and*
- *upon input of a tuple $\overline{a} \in V^{k-1}$ we can enumerate with constant delay and increasing order all nodes $b \in V$ such that $G \models \varphi(\overline{a}, b)$,*

Before turning to the proof of this proposition, let us first argue that the proof of Theorem 2.2 is completed by an easy application of the proposition: Let $\varphi(\overline{x}, x_k)$ be a first-order query of arity k and of q-rank at most ℓ. For a fixed $\epsilon > 0$, upon input of a $G \in C$ perform the preprocessing phase of the algorithm provided by Proposition 4.1 for the query $\varphi(\overline{x}, x_k)$. Furthermore, since $\psi(\overline{x}) := \exists x_k\, \varphi(\overline{x}, x_k)$ is a query of arity $k-1$, by our induction hypothesis the statement of Theorem 2.2 already holds for $\psi(\overline{x})$. Thus, after $O(|V|^{1+\epsilon})$ preprocessing time we can enumerate with constant

delay and in lexicographical order all $\overline{a} \in V^{k-1}$ such that $G \models \exists x_k\, \varphi(\overline{a}, x_k)$. For each such $\overline{a}$ we use the enumeration procedure provided by Proposition 4.1. This allows us to enumerate, with constant delay and lexicographical order, all tuples $(\overline{a}, b)$ with $G \models \varphi(\overline{a}, b)$.

The remainder of this section is devoted to the proof of Proposition 4.1.

To this end, consider the particular number $2kr$. For every $\lambda \in \mathbb{N}_{\geqslant 1}$ let C_λ be the subclass of C consisting of all colored graphs G such that Splitter wins the $(\lambda, 2kr)$-splitter game on the Gaifman graph of G. Clearly, $C_\lambda \subseteq C_{\lambda+1}$ for every $\lambda \in \mathbb{N}_{\geqslant 1}$. Since C is nowhere dense, by Theorem 3.7 there exists a number $\Lambda := \lambda(2kr) \in \mathbb{N}$ such that $C = C_\Lambda$. Thus,

$$C_1 \subseteq C_2 \subseteq \cdots \subseteq C_\Lambda = C.$$

We proceed by induction on λ such that $G \in C_\lambda$. The induction base for $\lambda = 1$ follows immediately, since by definition of the splitter game every $G \in C_1$ has to be edgeless, and thus a naive algorithm will work. For the induction step consider a $\lambda \geqslant 2$ and assume that the statement of Proposition 4.1 already holds for all colored graphs in $C_{\lambda-1}$.

We fix an $\epsilon' > 0$ and an FO$^+$-query $\varphi(\overline{x}, x_k)$ of arity k and q-rank at most ℓ. Our goal throughout the rest of this section is to provide an algorithm which upon input of a $G \in C_\lambda$ performs a preprocessing phase using time $O(|V|^{1+\epsilon'})$, such that afterwards

- upon input of a tuple $(\overline{a}, b) \in V^k$ we can test in constant time whether $G \models \varphi(\overline{a}, b)$, and
- upon input of a tuple $\overline{a} \in V^{k-1}$ we can enumerate with constant delay and increasing order all nodes $b \in V$ such that $G \models \varphi(\overline{a}, b)$.

Recall that the constant (delay or time) may depend on ϵ', φ, λ, k, q, ℓ, r, C, but not G.

When $G \models \varphi(\overline{a}, b)$ we say that b is a *matching solution* for $\overline{a}$ with respect to φ or, simply, a matching solution, when φ is clear from the context.

We first describe the preprocessing phase, then the testing procedure, and afterwards the enumeration procedure. While describing these, we also provide a runtime analysis and a correctness proof.

4.1 The preprocessing phase

Consider the query $\varphi(\overline{x}, x_k)$ with $\overline{x} = (x_1, \ldots, x_{k-1})$. Define $\epsilon := \epsilon'/k$.

Let $G \in C_\lambda$ be the input and let $n := |V|$ be the size of the domain V of G. The preprocessing phase is composed of the following steps:

1. For every $k' < k$ and every $\tau' \in \mathcal{T}_{k'}$, consider the k'-ary query $\rho_{\tau'}(x_1, \ldots, x_{k'})$ defined as the conjunction of the atoms $\mathrm{dist}(x_i, x_j) \leqslant r$ for all edges $\{i, j\}$ of τ' and the conjunction of the formulas $\mathrm{dist}(x_i, x_j) > r$ for all $i, j \in \{1, \ldots, k'\}$ with $i \neq j$ for which τ' does not contain the edge $\{i, j\}$. Note that for every $\overline{a} \in V^{k'}$ we have $G \models \rho_{\tau'}(\overline{a})$ iff $\tau' = \tau_r^G(\overline{a})$.
   Since $k' < k$, by our induction hypothesis, the statement of Theorem 2.2 holds for the query $\rho_{\tau'}$. Therefore, we can spend time $O(|V|^{1+\epsilon'})$ to perform the preprocessing phase provided by Theorem 2.2 for this query. Henceforth, for each $k' < k$ and each $\tau' \in \mathcal{T}_{k'}$, this will enable us upon input of a tuple $\overline{a} \in V^{k'}$, to test in constant time whether $G \models \rho_{\tau'}(\overline{a})$, i.e., whether $\tau_r^G(\overline{a}) = \tau'$.
2. Let $f_C(2kr, \epsilon)$ be the number provided by Theorem 3.3. If $n \leqslant f_C(2kr, \epsilon)$, we use a naive algorithm to compute the query result $\varphi(G)$ and trivially provide the functionality claimed by Proposition 4.1.
   From now on, consider the case where $n > f_C(2kr, \epsilon)$.
3. Using the algorithm provided by Theorem 3.3, we compute a $(kr, 2kr)$-neighborhood cover $\mathcal{X}$ of G with degree at most n^ϵ.
   Furthermore, in the same way as in [16, 17], we also compute for each $X \in \mathcal{X}$ a list of all $b \in V$ satisfying $\mathcal{X}(b) = X$, and we compute a node u_X such that $X \subseteq N^G_{2kr}(u_X)$.
   In addition, we use Lemma 3.5 to compute for every $X \in \mathcal{X}$ the r-kernel $K_r(X) = \{a \in X \,:\, N^G_r(a) \subseteq X\}$.
   All this can be done in time $O(n^{1+\epsilon'})$.
4. Let σ be the schema of G and use the algorithm provided by Theorem 3.9 upon input of $\varphi, G, \mathcal{X}$ to compute in time $O(n^{1+\epsilon'})$ the schema $\sigma^\star$, the $\sigma^\star$-expansion $G^\star$ of G, and for each $\tau \in \mathcal{T}_k$ the number m_τ, the FO$^+[\sigma^\star]$-sentences ξ^i_τ and the FO$^+[\sigma^\star]$-formulas $\psi^i_{\tau,I}(\overline{x}_I)$ of q-rank at most ℓ, for each $i \leqslant m_\tau$ and each connected component I of τ.
   Afterwards, we proceed in the same way as in [16, 17] to compute, within total time $O(n^{1+\epsilon})$, for every $X \in \mathcal{X}$ the structure $G^\star[X]$, and we let $G^\star_X$ be the expansion of $G^\star[X]$ where the new unary relation symbol K is interpreted by the r-kernel $K_r(X)$.
   Note that $G^\star_X$ has domain X and belongs to the class C_λ. Since $\mathcal{X}$ has degree at most n^ϵ, we have $\sum_{X \in \mathcal{X}} |X| \leqslant n^{1+\epsilon}$.
5. By the Rank-Preserving Normal Form Theorem 3.9 we have for all $\overline{a} = (a_1, \ldots, a_{k-1}) \in V^{k-1}$ and all $b = a_k \in V$ that $G \models \varphi(\overline{a}, b)$ if and only if there is a $\tau \in \mathcal{T}_k$ and an $i \leqslant m_\tau$ such that:
   (a) $\tau = \tau_r^G(\overline{a}, b)$
   (b) $G^\star \models \xi^i_\tau$
   (c) $G^\star[\mathcal{X}(b)] \models \psi^i_{\tau,J}(\overline{a}_J)$, where J is the connected component of τ with $k \in J$.
   Note that for $b = a_k$ the bag $\mathcal{X}(b)$ r-covers the tuple $\overline{a}_J$, since $k \in J$, J is a connected component of $\tau = \tau_r^G(\overline{a}, b)$, $|J| \leqslant k$, and $\mathcal{X}$ is a kr-neighborhood cover of G.
   (d) For all connected components I of τ with $k \notin I$ we have $G^\star[\mathcal{X}(\overline{a}_I)] \models \psi^i_{\tau,I}(\overline{a}_I)$, where $\mathcal{X}(\overline{a}_I)$ is defined to be $\mathcal{X}(a_{\min(I)})$; note that this bag r-covers $\overline{a}_I$.
   Using Theorem 3.1, we can test in time $O(|V|^{1+\epsilon})$ for every $\tau \in \mathcal{T}_k$ and $i \leqslant m_\tau$ whether $G^\star \models \xi^i_\tau$.
   We continue by performing the following preprocessing steps for all $\tau \in \mathcal{T}_k$ and every $i \leqslant m_\tau$ such that $G^\star \models \xi^i_\tau$. If there is no such τ and i then we can safely stop as there is no solution $(\overline{a}, b)$ with $G \models \varphi(\overline{a}, b)$.
6. For every connected component I of τ with $k \notin I$, the arity of the query $\psi^i_{\tau,I}(\overline{x}_I)$ is strictly smaller than k. By our induction hypothesis, the statement of Theorem 2.2 holds for this query. We let $\tilde{\epsilon} := \epsilon/(1+\epsilon)$. For each $X \in \mathcal{X}$ we use time $O(|X|^{1+\tilde{\epsilon}})$ to perform the preprocessing phase provided by Theorem 2.2 for the query $\psi^i_{\tau,I}(\overline{x}_I)$ and the colored graph $G^\star_X$. The total running time spent for this is of order at most

$$\sum_{X \in \mathcal{X}} |X|^{1+\tilde{\epsilon}} \leqslant \Big(\sum_{X \in \mathcal{X}} |X| \Big)^{1+\tilde{\epsilon}} \leqslant \big(n^{1+\epsilon}\big)^{1+\tilde{\epsilon}} \leqslant n^{1+\epsilon'}.$$

Here, the first inequality holds since $(x+y)^{1+\tilde{\epsilon}} = x(x+y)^{\tilde{\epsilon}} + y(x+y)^{\tilde{\epsilon}} \geqslant x x^{\tilde{\epsilon}} + y y^{\tilde{\epsilon}} = x^{1+\tilde{\epsilon}} + y^{1+\tilde{\epsilon}}$ is true for all reals $x, y, \tilde{\epsilon} \geqslant 0$; the second inequality is true since $\mathcal{X}$ has degree at most n^{ϵ}; and the third inequality is true since $\tilde{\epsilon} \leqslant \epsilon/(1+\epsilon)$ and $\epsilon \leqslant \epsilon'/k \leqslant \epsilon'/2$.

Having performed this preprocessing will henceforth enable us, upon input of a tuple $\overline{a}_I$ of elements in X, to test in constant time whether $G^\star[X] \models \psi^i_{\tau,I}(\overline{a}_I)$.

7. Let J be the connected component of τ with $k \in J$ and note that x_k is the last variable in the tuple $\overline{x}_J$. Let $z_1, \ldots, z_{k-|J|}$ be new variables and consider for each $p \in \{0, \ldots, k-|J|\}$ the query

$$\psi^i_{\tau,p}(z_1, \ldots, z_p, \overline{x}_J) \;:=\; \psi^i_{\tau,J}(\overline{x}_J) \;\wedge\; K(x_k) \;\wedge\; \varrho_\tau(\overline{x}_J) \;\wedge\; \bigwedge_{p' \in [1,p]} \mathrm{dist}(x_k, z_{p'}) > r\,.$$

Note that the query $\psi^i_{\tau,p}$ has q-rank at most ℓ.

The idea of this query is to make sure that $\psi^i_{\tau,J}(\overline{x}_J)$ is satisfied and that the nodes of $\overline{x}$ that are not in $\overline{x}_J$ but fall in the bag $\mathcal{X}(x_k)$ are sufficiently far away from x_k. Since we don't know in advance how many there will be, we anticipate all possibilities (by considering every $p \leqslant k-|J|$). Note that the arity of the query $\psi^i_{\tau,p}$ is k for $p = k-|J|$, and it is smaller than k for smaller p.

For every $X \in \mathcal{X}$ we would like to provide the following functionality: Upon input of a tuple of $p+|J|$ elements $c_1, \ldots, c_p, \overline{a}_J$ in X we want to be able to test in constant time whether $G^\star_X \models \psi^i_{\tau,p}(c_1, \ldots, c_p, \overline{a}_J)$. And upon input of a tuple of $p+|J|-1$ elements $c_1, \ldots, c_p, \overline{a}_{J\setminus\{k\}}$ we want to be able to enumerate with constant delay all $b = a_k$ in X such that

$$G^\star_X \models \psi^i_{\tau,p}(c_1, \ldots, c_p, \overline{a}_{J\setminus\{k\}}, b).$$

But as the query's arity $p+|J|$ might be as large as k, we do not have available the statement of Theorem 2.2 for this query. As a remedy, we perform the following steps 8–10 which make use of our second inductive assumption, stating that Proposition 4.1 already holds for the class $C_{\lambda-1}$ and for queries of arity up to k.

8. Recall that in step 3 we have already computed for every X in $\mathcal{X}$ a node u_X whose $2kr$-neighborhood contains X. Since $G \in C_\lambda$, we know that Splitter wins the $(\lambda, 2kr)$-splitter game on the Gaifman graph of G. For every $X \in \mathcal{X}$ we now compute a node v_X that is Splitter's answer if Connector plays u_X in the first round of the $(\lambda, 2kr)$-splitter game on the Gaifman graph of G. From [16] we know that the nodes $(v_X)_{X\in\mathcal{X}}$ can be computed within total time $O(n^{1+\epsilon'})$.
9. Let $p \in \{0, \ldots, k-|J|\}$, let $k' := p+|J|$ and let $\overline{z} = (z_1, \ldots, z_{k'}) := (z_1, \ldots, z_p, \overline{x}_J)$. Recall that $k \in J$ and x_k is the last variable of the tuple $\overline{x}_J$, hence $x_k = z_{k'}$. For every set $\overline{y}$ of variables from $\overline{z}$, we proceed as follows. For every $X \in \mathcal{X}$ we apply the Removal Lemma 3.8 to the colored graph $G^\star_X$, the query $\psi^i_{\tau,p}(\overline{z})$, the variables $\overline{y}$, and the node v_X. This yields a query $\psi'^i_{\tau,p,\overline{y}}(\overline{z} \setminus \overline{y})$ of q-rank at most ℓ and an expansion $H^\star_X$ of $G^\star_X \setminus \{v_X\}$ by unary predicates, such that for all k'-tuples $\overline{b}$ over X where $\{i \leqslant k' \,:\, b_i = v_X\} = \{i \leqslant k' \,:\, z_i \in \overline{y}\} =: I$ we have

$$G^\star_X \models \psi^i_{\tau,p}(\overline{b}) \quad\Longleftrightarrow\quad H^\star_X \models \psi'^i_{\tau,p,\overline{y}}(\overline{b}_{\setminus I}).$$

All this can be achieved in total time $O(n^{1+\epsilon'})$ (by a similar reasoning as in step 6).

10. By our choice of the node v_X we know that Splitter wins the $(\lambda-1, 2kr)$-splitter game on the Gaifman graph of $H^\star_X$. Hence, $H^\star_X$ belongs to $C_{\lambda-1}$, for every $X \in \mathcal{X}$. Using our induction hypothesis of Proposition 4.1 for $\tilde{\epsilon} := \epsilon/(1+\epsilon)$, we can thus spend total time at most $O(n^{1+\epsilon'})$ to perform for every $X \in \mathcal{X}$ the preprocessing phase that allows us to test with constant time and to enumerate with constant delay the results of queries on $H^\star_X$, provided that the queries have arity at most k and q-rank at most ℓ. Here, we carry this out for the queries $\psi'^i_{\tau,p,\overline{y}}(\overline{z} \setminus \overline{y})$, for all $\overline{y} \subseteq \overline{z}$.

Note that henceforth, this will allow us to do the following for every $X \in \mathcal{X}$:

- If $x_k \notin \overline{y}$, then when given an assignment $\overline{a}$ in $X \setminus \{v_X\}$ to the variables in $\overline{z} \setminus (\overline{y} \cup \{x_k\})$, we can enumerate with constant delay and in increasing order all assignments $b \in X \setminus \{v_X\}$ to the variable $x_k = z_{k'}$ such that $H^\star_X \models \psi'^i_{\tau,p,\overline{y}}(\overline{a}, b)$.
- If $x_k \notin \overline{y}$, then when given an assignment $\overline{a}$ in $X \setminus \{v_X\}$ to the variables in $\overline{z} \setminus (\overline{y} \cup \{x_k\})$ and an assignment b in $X \setminus \{v_X\}$ to the variable $x_k = z_k$, we can test in constant time whether $H^\star_X \models \psi'^i_{\tau,p,\overline{y}}(\overline{a}, b)$.
- If $x_k \in \overline{y}$, then when given an assignment $\overline{a}$ in $X \setminus \{v_X\}$ to the variables in $\overline{z} \setminus \overline{y}$, we can test in constant time whether $H^\star_X \models \psi'^i_{\tau,p,\overline{y}}(\overline{a})$.

The next two steps are only performed when $J = \{k\}$; otherwise the preprocessing stops here. Note that if $J = \{k\}$, then the tuple $\overline{x}_J$ only consists of the variable x_k. Furthermore, for a tuple $\overline{a} = (a_1, \ldots, a_{k-1})$ and $a_k = b$, the tuple $\overline{a}_J$ consists of the single element b.

11. Compute the set

$$L^i_{\tau,J} \;:=\; \{\, b \in V \,:\, G^\star[\mathcal{X}(b)] \models \psi^i_{\tau,J}(b) \,\}.$$

This can be achieved in total time $O(n^{1+\epsilon'})$ as follows. Choose $\tilde{\epsilon} := \epsilon/(1+\epsilon)$. For each $X \in \mathcal{X}$ use the algorithm provided by Theorem 3.1 to compute in time $O(|X|^{1+\tilde{\epsilon}})$ the result of the unary query $\psi^i_{\tau,J}$ on the colored graph $G^\star_X$, and let L_X be the intersection of this query result with the list of all elements b with $\mathcal{X}(b) = X$ (recall that we already precomputed this list in step 3). By the same reasoning as in step 6, the total time used for computing $(L_X)_{X\in\mathcal{X}}$ is at most $O(n^{1+\epsilon'})$. Furthermore, $L^i_{\tau,J}$ is the disjoint union of the sets L_X for all $X \in \mathcal{X}$.
12. Compute the *skip pointers* with respect to the set $L := L^i_{\tau,J}$ and $K_r(X)$ for all $X \in \mathcal{X}$ as in Lemma 3.10. By Lemma 3.10 this is done in time $O(n^{1+k\epsilon}) = O(n^{1+\epsilon'})$.

This concludes the preprocessing phase. Note that the total time spent by this preprocessing phase is $O(n^{1+\epsilon'})$, as desired.

4.2 Testing

We now describe how, upon input of a tuple $(\overline{a}, b) \in V^k$, we can test in constant time whether $G \models \varphi(\overline{a}, b)$. Let $(a_1, \ldots, a_{k-1}) = \overline{a}$ and $a_k = b$. From step 5 of the preprocessing phase we know that $G \models \varphi(\overline{a}, b)$ if and only if there exists a $\tau \in \mathcal{T}_k$ and an $i \leqslant m_\tau$ such that:

(1) $\tau = \tau_r^G(\overline{a}, b)$,
(2) $G^\star \models \xi_\tau^i$,
(3) $G^\star[\mathcal{X}(b)] \models \psi^i_{\tau,J}(\overline{a}_J)$ where J is the connected component of τ with $k \in J$, and
(4) for all connected components I of τ with $k \notin I$ we have $G^\star[\mathcal{X}(\overline{a}_I)] \models \psi^i_{\tau,I}(\overline{a}_I)$, where $\mathcal{X}(\overline{a}_I)$ denotes the bag $\mathcal{X}(a_{\min(I)})$.

Therefore, it suffices to check for each $\tau \in \mathcal{T}_k$ and each $i \leqslant m_\tau$ whether the conditions (1)–(4) are satisfied. Note that the number of relevant (τ, i) is a constant, so we can check each choice of (τ, i) separately. For each fixed $\tau \in \mathcal{T}_k$ and $i \leqslant m_\tau$, we proceed as follows.

The items (2) and (4) are easy to deal with using the information precomputed in steps 5 and 6 of the preprocessing phase. If at least one of them is false, we can stop as $(\overline{a}, b)$ is not a solution for this (τ, i). Otherwise, we proceed to check the items (1) and (3).

To this end, let τ' be the subgraph of τ induced on the set $\{1, \dots, k-1\}$, and use the functionality provided in step 1 of the preprocessing phase to test in constant time whether $\tau_r^G(a_1, \dots, a_{k-1}) = \tau'$. If this test fails, we know that item (1) cannot be true, and hence we can stop as $(\overline{a}, b)$ is not a solution for this (τ, i). Otherwise, proceed as follows.

Let J be the connected component of τ with $k \in J$. Recall that the tuple $\overline{a}_J$ consists of all components of the tuple $(a_1, \dots, a_{k-1}, a_k)$ whose indices belong to J. In particular, the last entry of the tuple $\overline{a}_J$ is $a_k = b$. We define:

- $X := \mathcal{X}(b)$. Recall that X r-covers the tuple $\overline{a}_J$. Therefore, for every $w \in \{a_1, \dots, a_k\} \setminus X$ we know that $G \models \mathrm{dist}(w, a_k) > r$.
- Let p be the number of elements in $\{a_1, \dots, a_k\}$ that belong to X but that do not occur in the tuple $\overline{a}_J$, and let $c_1, \dots, c_p$ be a list of all these elements.
  In order to check item (1), we have to make sure that a_k is far from each of the nodes $c_1, \dots, c_p$.
- Let $\overline{y}$ consist of the variables x_ν for all $\nu \in \{1, \dots, k\}$ such that $a_\nu = v_X$.
- Let $c'_1, \dots, c'_{p'}$ be the elements of $c_1, \dots, c_p$ that are not equal to v_X.
- Let $\overline{a}'_J$ be the tuple obtained from $\overline{a}_J$ by removing all components whose entry is v_X.

Consider the formula $\psi^i_{\tau,p}$ from step 7 of the preprocessing phase. Note that the items (1) and (3) are satisfied if and only if $G^\star_X \models \psi^i_{\tau,p}(c_1, \dots, c_p, \overline{a}_J)$. By the Removal Lemma 3.8, the latter is satisfied iff $H^\star_X \models \psi'^i_{\tau,p,\overline{y}}(c'_1, \dots, c'_{p'}, \overline{a}'_J)$ — and this can be checked in constant time by using the functionality provided by the second and the third bullet in step 10 of the preprocessing phase.

In summary, when given a tuple $(\overline{a}, b)$, we only spend constant time to test whether $G \models \varphi(\overline{a}, b)$.

4.3 Enumeration

We now describe how, upon input of a tuple $\overline{a} = (a_1, \dots, a_{k-1}) \in V^{k-1}$ we can enumerate with constant delay and increasing order all nodes $b = a_k \in V$ such that $G \models \varphi(\overline{a}, b)$. What we actually do is the following: For a given pair (τ, i), we enumerate in lexicographical order all b such that the items (1)–(4) listed at the beginning of Section 4.2 are satisfied.

As there are only a constant number of pairs (τ, i), we will enumerate all corresponding solutions concurrently and use this to obtain a global enumeration in lexicographical order with constant delay.

We now consider a fixed pair (τ, i). Let τ' be the subgraph of τ induced on $\{1, \dots, k-1\}$, and use the functionality provided by step 1 of the preprocessing phase to test in constant time whether $\tau_r^G(\overline{a}) = \tau'$. If this is not the case, we know that for this τ, the condition of item (1) cannot be satisfied by any $b \in V$. Therefore, we can safely enumerate the empty set.

Otherwise, i.e., if $\tau_r^G(\overline{a}) = \tau'$, we proceed as follows.

By step 5 of the preprocessing phase, we can look up in constant time whether $G^\star \models \xi_\tau^i$. We thus know if item (2) is satisfied. Furthermore, using the functionality provided in step 6 of the preprocessing phase, we can test in constant time for all connected components I of τ with $k \notin I$, whether $G^\star[\mathcal{X}(\overline{a}_I)] \models \psi^i_{G,I}(\overline{a}_I)$. Afterwards, we know if item (4) is satisfied.

If one of the items (2) or (4) is not satisfied, we know that there is no matching solution for this $\overline{a}$ and (τ, i), and we can therefore safely enumerate the empty set.

Otherwise, i.e., if the items (2) and (4) are satisfied, we let J be the connected component of τ with $k \in J$, and we proceed with the two following cases.

Case I: $J = \{k\}$.

In this case, every matching solution b for this $\overline{a}$ and this (τ, i) has to be of distance $> r$ to every element in $\overline{a}$. Consider the bags $\mathcal{X}(a_1), \dots, \mathcal{X}(a_{k-1})$, let $k' := |\{\mathcal{X}(a_\nu) : \nu \in \{1, \dots, k-1\}\}|$, and let $X_1, \dots, X_{k'}$ be a list of these bags. Clearly, $k' \leqslant k-1$, and for each component a_ν of $\overline{a}$, there is exactly one κ such that $\mathcal{X}(a_\nu) = X_\kappa$.

For each $\kappa \leqslant k'$, we

- let p_κ be the number of elements in $\{a_1, \dots, a_{k-1}\}$ that belong to X_κ, and we let $\overline{c}_\kappa := (c_{\kappa,1}, \dots, c_{\kappa,p_\kappa})$ be a list of all these elements.
- Let $\overline{y}$ consist of the variables x_ν for all $\nu \in \{1, \dots, k-1\}$ such that $a_\nu = v_{X_\kappa}$.
- Let $\overline{c}'_\kappa = (c'_{\kappa,1}, \dots, c'_{\kappa,p'_\kappa})$ be a list of all elements of $\overline{c}_\kappa$ that are not equal to v_{X_κ}.

We execute simultaneously in parallel the following $2k' + 1$ enumeration procedures:

- For each $\kappa \in \{1, \dots, k'\}$ we want to enumerate all $b \in X_\kappa \setminus \{v_{X_\kappa}\}$ that are far away from all the nodes in the list $\overline{c}_\kappa$. More precisely, we want to enumerate all $b \in X_\kappa \setminus \{v_{X_\kappa}\}$ such that $G^\star_{X_\kappa} \models \psi^i_{\tau,p_\kappa}(\overline{c}_\kappa, b)$. Note that these are exactly all the nodes $b \in K_r(X_\kappa) \setminus \{v_{X_\kappa}\}$ that satisfy the items (1) and (3).
  From the statement made at the end of step 9 of the preprocessing phase, we know that to enumerate all these nodes b, we can use the functionality provided by the first bullet of step 10 of the preprocessing phase: We enumerate, with constant delay and in increasing order, all $b \in X_\kappa \setminus \{v_{X_\kappa}\}$ such that $H^\star_{X_\kappa} \models \psi'^i_{\tau,p_\kappa,\overline{y}}(\overline{c}'_\kappa, b)$.
- For each $\kappa \in \{1, \dots, k'\}$ we want to check if $G^\star_{X_\kappa} \models \psi^i_{\tau,p_\kappa}(\overline{c}_\kappa, b)$ holds for the particular node $b := v_{X_\kappa}$, and if so, we want to output this node (otherwise, we enumerate the empty set).

By the statement made at the end of step 9 of the preprocessing phase, this check can be performed by using the functionality provided by the third bullet of step 10 of the preprocessing phase: We simply check in constant time if $H^\star_{X_\kappa} \models \psi'^i_{\tau,p_\kappa,\overline{y}\cup\{x_k\}}(\overline{c}'_\kappa)$.

- Using the functionality provided by step 12 of the preprocessing phase, we enumerate with constant delay and in increasing order the set

$$\{\, b \in L \;:\; b \notin \bigcup_{\kappa \leqslant k'} K_r(X_\kappa) \,\}$$

where $L := L^i_{\tau,J}$ is the set computed in step 11 of the preprocessing phase.

It should be clear that every b that is enumerated by this process is a matching solution for $\overline{a}$ and (τ, i). Let us now argue that our enumeration process produces *all* matching solutions for $\overline{a}$ and (τ, i): Note that any matching solution b for $\overline{a}$ and (τ, i) is either in the canonical bag $\mathcal{X}(a_\nu)$ of one of the elements a_ν in $\overline{a}$, or it is outside all these bags. In the first case, b is produced by one of the enumeration procedures of the first two bullets. In the second case, b is enumerated by the skip pointers. Therefore, all matching solutions b will eventually be enumerated.

Case II: $\{k\} \subsetneq J$

W.l.o.g. let us assume that $1 \in J$ and that $\{1, k\}$ is an edge in τ.

Regarding item (3), note that the Rank-Preserving Normal Form Theorem 3.9 tells us that instead of the bag $\mathcal{X}(b)$ we can use *any* bag X that r-covers $\overline{a}_J$.

We define:

- $X := \mathcal{X}(a_1)$. Note that every $b \in V$ that satisfies item (1) belongs to X and, moreover, X r-covers $\overline{a}_J$ for $\overline{a} = (a_1, \ldots, a_{k-1})$ and $a_k := b$.
- Let p be the number of elements of $\{a_1, \ldots, a_{k-1}\}$ that belong to X but not to the tuple $\overline{a}_J$. Let $c_1, \ldots, c_p$ be the list of all these elements.
- Let $c'_1, \ldots, c'_{p'}$ be the elements of $c_1, \ldots, c_p$ that are not equal to v_X.
- Let $\Gamma := J \setminus \{k\}$.
- Let $\overline{a}'_\Gamma$ be the tuple obtained from $\overline{a}_\Gamma$ by removing all components whose entry is v_X.
- Let $\overline{y}$ consist of the variables x_ν for all $\nu \in \{1, \ldots, k-1\}$ such that $a_\nu = v_X$.

Since all matching b must be close to a_1 in this case, it suffices to run in parallel the following two enumeration procedures:

- We want to enumerate all $b \in X \setminus \{v_X\}$ that are far from all the nodes $c_1, \ldots, c_p$. More precisely, we want to enumerate all $b \in X \setminus \{v_X\}$ such that $G^\star_X \models \psi^i_{\tau,p}(c_1, \ldots, c_p, \overline{a}_\Gamma, b)$. Note that these are precisely the nodes $b \in V \setminus \{v_X\}$ that satisfy the items (1) and (3).
  From the statement made at the end of step 9 of the preprocessing phase, we know that to enumerate all these nodes b, we can use the functionality provided by the first bullet of step 10 of the preprocessing phase: We enumerate, with constant delay and in increasing order, all $b \in X \setminus \{v_X\}$ such that $H^\star_X \models \psi'^i_{\tau,p,\overline{y}}(c'_1, \ldots, c'_{p'}, \overline{a}'_\Gamma, b)$.
- We want to check if $G^\star_X \models \psi^i_{\tau,p}(c_1, \ldots, c_p, \overline{a}_\Gamma, b)$ holds for the particular node $b := v_X$, and if so, we want to output this node (otherwise, we enumerate the empty set).
  From the statement made at the end of step 9 of the preprocessing phase, we know that this check can be performed by using the functionality provided by the third bullet of step 10 of the preprocessing phase: We simply check in constant time if $H^\star_X \models \psi'^i_{\tau,p,\overline{y}\cup\{x_k\}}(c'_1, \ldots, c'_{p'}, \overline{a}'_\Gamma)$.

This concludes the description of the enumeration procedure. While describing this procedure, we have already verified that it outputs exactly those $b \in V$ for which $G \models \varphi(\overline{a}, b)$.

As we execute concurrently a constant number of enumeration processes and since all of them produce solutions in increasing order and with constant delay, we can ensure that no tuple is produced twice even if several sub-procedures share some solutions. This completes the proof of Proposition 4.1 and hence also completes the proof of Theorem 2.2.

5 CONCLUSION

We have shown how to efficiently enumerate the results of first-order queries over any nowhere dense class of databases. We achieved constant delay enumeration after a pseudo-linear time preprocessing. We also showed that after a pseudo-linear preprocessing we can, on input of an arbitrary tuple, test in constant time whether it is a solution to the query.

We did not mention the size of the constant factor. Already for boolean queries 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 if the class of structures contains all trees (unless FPT = AW[*], cf. [14]).

An improvement of our work would be to extend the results in a dynamic setting that avoids recomputing from scratch the index built during the preprocessing phase. For instance, the index structure allowing for constant delay enumeration can be updated in constant time in the setting of FO queries over classes of databases of bounded degree [8] and in the setting of q-hierarchical unions of conjunctive queries over arbitrary databases [7, 9]. In the nowhere dense case, constant update time seems unrealistic, as already for boolean queries over trees the best we can do so far are logarithmic time updates [6].

It seems plausible that there exists an index structure, computable in pseudo-linear time and allowing for constant delay enumeration and logarithmic time updates. Preliminary results were obtained in this direction for very simple structures such as words [27]. Generalisation to more complex structures remains for future work.

REFERENCES

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

[2] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear cost sequential and distributed constructions of sparse neighborhood covers. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 638–647. IEEE Computer Society, 1993.

[3] Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 503–513. IEEE Computer Society, 1990.

[4] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary,*

September 25-29, 2006, Proceedings, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[5] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007.

[6] Andrey Balmin, Yannis Papakonstantinou, and Victor Vianu. Incremental validation of XML documents. *ACM Trans. Database Syst.*, 29(4):710–751, 2004.

[7] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318. ACM, 2017.

[8] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[9] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under updates and in the presence of integrity constraints. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, volume 98 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[10] Rodney G. Downey, Michael R. Fellows, and Udayan Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. In Douglas S. Bridges, Cristian S. Calude, Jeremy Gibbons, Steve Reeves, and Ian H. Witten, editors, *First Conference of the Centre for Discrete Mathematics and Theoretical Computer Science, DMTCS 1996, Auckland, New Zealand, December, 9-13, 1996*, pages 194–213. Springer-Verlag, Singapore, 1996.

[11] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4):21, 2007.

[12] Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 121–131. ACM, 2014.

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

[14] 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.

[15] Haim Gaifman. On local and non-local properties. In *Proceedings of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105 – 135. Elsevier, 1982.

[16] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 89–98. ACM, 2014.

[17] Martin Grohe and Nicole Schweikardt. First-order query evaluation with cardinality conditions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018, Houston, TX, USA, June 10–15, 2018*. ACM, 2018. Full version available at CoRR, http://arxiv.org/abs/1707.05945, 2017.

[18] 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.

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

[20] Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 297–308. ACM, 2013.

[21] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25:1–25:12, 2013.

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

[23] Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[24] Jaroslav Nesetril and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *J. Symb. Log.*, 75(3):868–887, 2010.

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

[26] Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics.* Springer, 2012.

[27] Matthias Niewerth and Luc Segoufin. Enumeration of MSO queries on strings with constant delay and logarithmic updates. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018, Houston, TX, USA, June 10–15, 2018.* ACM, 2018.

[28] David Peleg. Distance-dependent distributed directories. *Inf. Comput.*, 103(2):270–298, 1993.

[29] Luc Segoufin and Alexandre Vigny. Constant delay enumeration for FO queries over databases with local bounded expansion. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[30] Sebastian Siebertz. *Nowhere Dense Classes of Graphs: Characterisations and Algorithmic Meta- Theorems.* PhD thesis, Technical University of Berlin, Germany, 2016.

[31] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982.

APPENDIX

In this appendix, we present three examples that highlight the importance of specific information that is computed during the preprocessing phase described in Section 4.1.

In our example queries, we write $\mathrm{dist}((y,z);x) > r$ as a shorthand for the conjunction of the distance atoms $\mathrm{dist}(y,x) > r$ and $\mathrm{dist}(z,x) > r$. The variable y will play the role of the variable x_k in Section 4.1, where k is the arity of the considered query.

First example. Our first example highlights the importance of using the r-kernels $K_r(X)$ for the bags X of a neighborhood cover $\mathcal{X}$.

Consider the query $\varphi(x,y)$ that returns all pairs of nodes (x,y) that are far away from each other and for which there is a blue node close to y but far from x. I.e., $\varphi(x,y) =$

$$\exists z \left(B(z) \wedge \mathrm{dist}(y,z) < r \wedge \mathrm{dist}((y,z);x) > r \right).$$

Assume that during a preprocessing phase we have computed a $3r$-neighborhood cover $\mathcal{X}$ and the set $L := \left\{ b \in V \;:\; \exists z \left(B(z) \wedge \mathrm{dist}(b,z) < r \right) \right\}$.

Now, let us assume we are given a tuple (a,b), and we want to test if this tuple is a solution to the query.

In case that $b \notin \mathcal{X}(a)$, we know that $\mathrm{dist}(a,b) > 3r$, and hence (a,b) is a solution to the query if and only if $b \in L$. Thus, for $b \notin \mathcal{X}(a)$, testing if $G \models \varphi(a,b)$ boils down to testing if $b \in L$.

However, in case that $b \in \mathcal{X}(a)$, we won't be able to conclude with just the information on $\mathcal{X}$ and L, regardless of any further precomputation on $\mathcal{X}(a)$, because the blue node z that's close to b and far from a (and hence witnesses that (a,b) is a solution to the query) might be outside the bag $\mathcal{X}(a)$; see Figure 1 for an illustration.

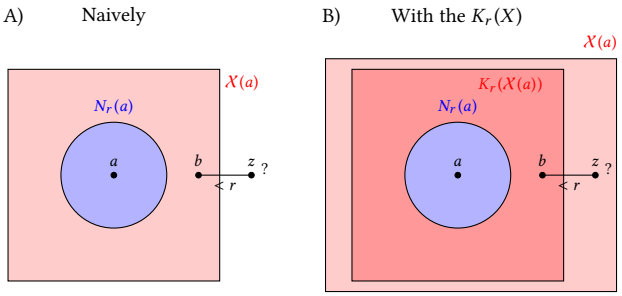
To solve this issue, we use the r-kernels $K_r(X)$ for all bags $X \in \mathcal{X}$ and make sure that these are computed during the preprocessing phase as well. Then, for a given tuple (a,b) we know the following.

If $b \in K_r(\mathcal{X}(a))$, then $N_r^G(a,b) \subseteq \mathcal{X}(a)$, and hence, $G \models \varphi(a,b) \iff G[\mathcal{X}(a)] \models \varphi(a,b)$, and a suitable precomputation within $\mathcal{X}(a)$ can yield the desired result.

On the other hand, if $b \notin K_r(\mathcal{X}(a))$ then, since $N_{3r}^G(a) \subseteq \mathcal{X}(a)$, we have $G \models \mathrm{dist}(a,b) > 2r$, and therefore, $G \models \varphi(a,b) \iff b \in L$.

Second example. Our second example highlights the importance of the queries $\psi_{\tau,p}^i$ defined in step 7 of the preprocessing phase. We

Figure 1: The use of $K_r(X)$



consider the query $\varphi(x_1, x_2, x_3, y) :=$

$$E(x_1, y) \ \wedge \bigwedge_{i \in \{2,3\}} \mathrm{dist}((x_1, y); x_i) > r$$

Assume that during a preprocessing phase we have computed a kr-neighborhood cover $\mathcal{X}$ for some number $k \geqslant 2$. In some cases, this is enough for our needs: If we are given a tuple (a_1, a_2, a_3, b) with $a_2 \notin \mathcal{X}(a_1)$ and $a_3 \notin \mathcal{X}(a_1)$, then $G \models \varphi(a_1, a_2, a_3, b)$ if and only if $G \models E(a_1, b)$, and this can be checked easily.

But if a_2 or a_3 (or both) falls into $\mathcal{X}(a_1)$, we need something more. The problem is that we don't know in advance which situation will occur. Fortunately, there is only a constant number of possibilities, and we can perform an adequate preprocessing for each of them. Each situation is associated to one of the following queries

$$\begin{aligned} \psi_1(z_1, x_1, y) &:= E(x_1, y) \wedge \mathrm{dist}((x_1, y); z_1) > r \\ \psi_2(z_1, z_2, x_1, y) &:= E(x_1, y) \wedge \\ &\qquad \bigwedge_{i \in \{1,2\}} \mathrm{dist}((x_1, y); z_i) > r\,, \end{aligned}$$

and these correspond to the queries $\psi^i_{\tau,p}$ in step 7 of the preprocessing phase.

We now have that $G \models \varphi(a_1, a_2, a_3, b)$ if and only if one of the following is true:

- $a_2 \notin \mathcal{X}(a_1)$ & $a_3 \notin \mathcal{X}(a_1)$ and $G[\mathcal{X}(a_1)] \models E(a_1, b)$
- $a_2 \notin \mathcal{X}(a_1)$ & $a_3 \in \mathcal{X}(a_1)$ and $G[\mathcal{X}(a_1)] \models \psi_1(a_3, a_1, b)$
- $a_2 \in \mathcal{X}(a_1)$ & $a_3 \notin \mathcal{X}(a_1)$ and $G[\mathcal{X}(a_1)] \models \psi_1(a_2, a_1, b)$
- $a_2 \in \mathcal{X}(a_1)$ & $a_3 \in \mathcal{X}(a_1)$ and $G[\mathcal{X}(a_1)] \models \psi_2(a_2, a_3, a_1, b)$

In each of these cases, we made progress as we can now restrict our attention to one bag and do not have to consider the entire graph.

Third example. Our last example illustrates the use of the Rank-Preserving Normal Form Theorem 3.9. Consider the query

$$\varphi(y) := \forall z \Big((\, \mathrm{dist}(y, z) \leqslant 2 \ \wedge \ B(z)\,) \to E(y, z)\,) \Big).$$

When applying the Removal Lemma 3.8, we get the query

$$\begin{aligned} \varphi'(y) := \quad & \forall z \Big((\, \mathrm{dist}(y, z) \leqslant 2 \wedge B(z)\,) \to E(y, z)\,) \Big) \ \wedge \\ & \forall z \Big((\, R_1(z) \wedge R_1(y) \wedge B(z)\,) \to E(y, z) \Big) \ \wedge \\ & \Big((\, R_2(y) \wedge \xi\,) \to R_1(y)\,) \Big). \end{aligned}$$

Here, for $d \in \{1, 2\}$, R_d is a new unary predicate such that $R_d(z)$ means that z is a node at distance d from the removed node, and ξ is true when the removed node is blue, and ξ is false otherwise.

Note that in the second line of the new formula we lose control over the quantification on z: After removing a node, the distance between two elements (distance 2 in this example) may be arbitrarily large.

The second line of the formula, when put in Gaifman Normal Form, yields a formula which expresses the following: it first checks for the existence of two nodes z and z' that satisfy R_1 and B and are at distance strictly more than two from each other. If this is the case, we are sure that y does not belong to the query result, because one of the nodes z, z' is clearly not connected to y.

Otherwise, i.e., if this is not the case, we know the following: If there is a node satisfying B and R_1 in the 2-neighborhood of y, then all nodes z satisfying B and R_1 are in the 4-neighborhood of y, so it is enough to check this neighborhood. But note that the radius we have to consider has increased.

In the same situation, using the Rank-Preserving Normal Form Theorem 3.9 instead of Gaifman's Theorem does *not* increase the radius. But applying Theorem 3.9 requires to enrich the colored graph by further unary predicates, and this can be done efficiently only in the nowhere dense case.