On the Complexity of Enumerating the Answers to Well-designed Pattern Trees

Markus Kröll[1], Reinhard Pichler[2], and Sebastian Skritek[3]

1 **TU Wien, Vienna, Austria**
kroell@dbai.tuwien.ac.at
2 **TU Wien, Vienna, Austria**
pichler@dbai.tuwien.ac.at
3 **TU Wien, Vienna, Austria**
skritek@dbai.tuwien.ac.at

Abstract

Well-designed pattern trees (wdPTs) have been introduced as an extension of conjunctive queries to allow for partial matching – analogously to the OPTIONAL operator of the semantic web query language SPARQL. Several computational problems of wdPTs have been studied in recent years, such as the evaluation problem in various settings, the counting problem, as well as static analysis tasks including the containment and equivalence problems. Also restrictions needed to achieve tractability of these tasks have been proposed. In contrast, the problem of enumerating the answers to a wdPT has been largely ignored so far. In this work, we embark on a systematic study of the complexity of the enumeration problem of wdPTs. As our main result, we identify several tractable and intractable cases of this problem both from a classical complexity point of view and from a parameterized complexity point of view.

1998 ACM Subject Classification H.2.3 Database Management – Languages

Keywords and phrases SPARQL, Pattern Trees, CQs, Enumeration, Complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.22

1 Introduction

With the steadily increasing amount of inaccurate and incomplete data on the web, the need for partial matching as an extension of Conjunctive Queries (CQs) is gaining more and more importance. Therefore, in the semantic web query language SPARQL, the OPTIONAL operator is a crucial feature. Recently, this operator (which we shall abbreviate to "OPT" in the sequel) has also been studied for arbitrary relational vocabulary [4]. Intuitively, the OPT operator (which corresponds to the left outer join in the Relational Algebra) allows the user to extend CQs by optional parts which are retrieved from the data if available, but which do not cause the partial answers to get lost in case the optional information is not available. The following example will help to illustrate this idea.

▶ **Example 1.** Consider the following {AND,OPT}-SPARQL query that is posed over a database that stores information about movies:[1]

$$\Big(\big((x, \texttt{directed\_by}, y)\ \mathsf{AND}\ (x, \texttt{released}, \text{“before\_1980”})\big)\ \mathsf{OPT}\ (x, \texttt{oscars\_won}, z)\Big)\ \mathsf{OPT}\ (y, \texttt{first\_movie}, z').$$

[1] We use here the algebraic-style notation from [25] rather than the official SPARQL syntax of [28].

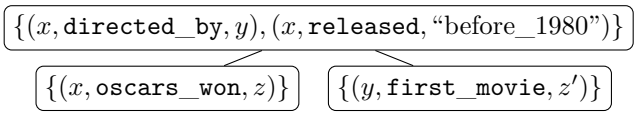

Figure 1 The wdPT representing the query from Example 1.

This query retrieves all pairs (m, d) s.t. movie m is directed by d and released before 1980. This is specified by the pattern $(x, \texttt{directed\_by}, y)$ AND $(x, \texttt{released}, \text{“before\_1980”})$. Furthermore, whenever possible, this query also retrieves (one or both of) the following pieces of data: the number of Academy Awards n won by movie m and the first movie m' directed by d. In other words, in addition to (m, d) we also retrieve n and/or m' if the information is available in the database. This is specified by the atoms $(x, \texttt{oscars\_won}, z)$ and $(y, \texttt{first\_movie}, z')$ following the respective OPT-operators.

In [23], so-called *well-designed pattern trees* (wdPTs) were introduced as a convenient graphical representation of CQs extended by the OPT operator. Intuitively, the nodes in a wdPT correspond to CQs while the tree structure represents the optional extensions. The wdPT corresponding to the query in Example 1 is displayed in Figure 1.

The semantics of a wdPT p is defined in terms of the CQs contained in it: with each subtree T' of T containing the root, we associate a CQ $q_{T'}$ defined by the conjunction of all atoms in the nodes of T'. The evaluation of wdPT p over database $\mathcal{D}$ consists then of all “maximal” answers to the CQs of the form $q_{T'}$. That is, we take the union of all answers to the CQs of the form $q_{T'}$, for each T', and then remove all those answers that are “extended” by some other answer in the set. We revisit Example 1 to illustrate these ideas.

▶ **Example 2.** Consider an RDF database $\mathcal{D}$ consisting of triples (“American_Graffiti”, `directed_by`, “George_Lucas”), (“American_Graffiti”, `released`, “before_1980”), (“Star_Wars”, `directed_by`, “George_Lucas”), (“Star_Wars”, `released`, “before_1980”), (“Star_Wars”, `oscars_won`, “6”). The evaluation over $\mathcal{D}$ of the wdPT in Figure 1, and, therefore, of the query from Example 1, consists of partial mappings μ_1 and μ_2 defined on variables x, y, z, z' such that: (1) μ_1 is only defined on x and y in such a way that $\mu_1(x) =$ “American_Graffiti” and $\mu_1(y) =$ “George_Lucas”, and (2) μ_2 is defined on x, y and z in such a way that $\mu_2(x) =$ “Star_Wars”, $\mu_2(y) =$ “George_Lucas”, and $\mu_2(z) =$ “6”.

To make wdPTs a proper extension of CQs, wdPTs have to be enhanced with projection. For instance, for the wdPT in Example 1, one might decide to project out the variable x. In this way, the answers μ_1 and μ_2 over the database in Example 2 would be restricted to μ_1' and μ_2', s.t. (1) μ_1' is only defined on y with $\mu_1'(y) =$ “George_Lucas”, and (2) μ_2' is defined on y and z and it holds that $\mu_2'(y) =$ “George_Lucas” and $\mu_2'(z) =$ “6”.

Several aspects of the complexity of wdPTs have been studied in previous works. Given a database $\mathcal{D}$ and a query p, the evaluation problem asks if some given mapping μ is an answer to p over $\mathcal{D}$. This problem was shown coNP-complete for wdPTs without projection [25] and Σ_2^P-complete for wdPTs with projection [23]. Wrapping wdPTs into the CONSTRUCT operator of SPARQL makes the evaluation problem NP-complete [20]. In [2], the max-evaluation problem was introduced as a variant of the evaluation problem. Note that if we allow projection, then it may happen that both, some mapping μ and a proper extension of μ are solutions. This is indeed the case for μ_1' and μ_2' above. In [2], the *max-evaluation* problem was identified as an important variant of wdPT evaluation: it asks if a given mapping is a *maximal* solution (i.e., it cannot be properly extended to another solution). This problem is DP-complete [2]. In [27], the counting problem of wdPTs was studied, i.e., given a database

$\mathcal{D}$ and a query p, what is the number of solutions? It turned out that the counting problem of wdPTs remains intractable even under very severe restrictions. Important query analysis tasks such as the containment and equivalence problems in various settings were studied in [26]. It was shown that the complexity of these tasks ranges from NP-completeness (if we disallow projection) to undecidability (if projection is allowed in both queries).

Another interesting and very natural problem in the context of query languages is the *enumeration problem* (i.e., computing one solution after the other without ever outputting duplicates). Despite recent interest in the enumeration problem for First-Order and Conjunctive Queries [5, 19, 29, 12], the complexity of enumerating the answers to wdPTs has been hardly considered so far. A notable exception is [23], where it was shown that enumeration of wdPTs without projection is tractable provided that the CQs contained in the wdPTs are from some tractable fragment. In contrast, for wdPTs with projection, the enumeration problem of wdPTs was shown intractable in [23] even when allowing only acyclic CQs to appear at each node of a wdPT.

Goal. The goal of this work is to initiate a systematic complexity study of the enumeration problem of wdPTs. We thus view the enumeration problem from various angles:

- *Classical complexity analysis of the combined complexity.* We aim at identifying the boundary between tractable and intractable enumeration for the set of both, *all* solutions and *maximal* solutions. In [17], various notions of tractable enumeration are defined. We concentrate on two such notions, namely polynomial delay as the strongest and output polynomial time as the weakest form of tractability. The former means that the time before outputting the first solution, the time between any two solutions and the time between the last output and termination are all bounded by a polynomial in the size of the input. The latter means that the total time for outputting all solutions is bounded by a polynomial in the combined size of the input *plus* the output. In our quest for tractable enumeration, we start with restrictions introduced in [4] to achieve tractable evaluation and we will introduce further restrictions to get a better understanding of the sources of complexity.
- *Parameterized complexity.* Over more than a decade, parameterized complexity theory [11] has developed into a well-established approach to dealing with intractability. The ideal result of a parameterized complexity analysis is fixed-parameter tractability. This means that the problem at hand can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the size of the input and $f(k)$ is a function depending solely on the parameter k. In other words, the exponential explosion can thus be confined to the parameter. By FPT, we denote the class of problems with this behavior. The opposite behavior is fixed-parameter intractability. This means that we can at best achieve an upper bound $\mathcal{O}(n^{f(k)})$ on the time complexity. In other words, the size of the parameter occurs in the exponent of the input size n. In this paper, we take the size of the query as parameter. In a sense, this gives us an intermediate state between data complexity (where the query is considered as fixed) and classical combined complexity (where the query is treated equally as the data). The parameterized complexity approach allows us to obtain a more fine-grained picture of the intractable cases by exploring the boundary between FPT-delay and fixed-parameter intractable enumeration.
- *Revisiting the evaluation problem.* Our complexity analysis of the enumeration problem will bring to light an interesting relationship between fixed-parameter tractable *enumeration* and *evaluation*. More formally, we will show that FPT-delay of the enumeration problem for some type of wdPTs implies that also the evaluation problem for this type

of wdPTs is in FPT. We thus resume the quest for tractable evaluation from [4] and inspect the intractable cases (classical complexity) from a parameterized complexity point of view. We thus aim at delineating the border between fixed-parameter tractability and fixed-parameter intractability to get a better understanding of the nature of the intractability.

- *Data complexity.* Finally, we have a brief look at the data complexity of the enumeration problem of wdPTs. Note that polynomial delay is trivial for query languages derived from First-Order logic. Hence, the goal of analyzing the data complexity is to construct an enumeration algorithm which works with linear-time preprocessing (i.e., the time to produce the first output) and constant delay (i.e., the time between any two successive outputs and the time after the last output).

Organization of the paper and summary of results. In Section 2, we recall some basic notions and results. A conclusion and an outlook to future work are given in Section 6. The main results of the paper are detailed in Sections 3–5, namely:

- *Evaluation problem.* In Section 3, we revisit the evaluation problem of wdPTs. More precisely, we subject the intractable cases from [4] for the evaluation problem to a parameterized complexity analysis. By establishing both fixed-parameter tractability and intractability results, we provide a more fine-grained picture of the complexity of this problem.
- *Combined complexity.* In Section 4, we study the combined complexity of enumerating *all* solutions and of enumerating the *maximal* solutions. We have already recalled above that – without any restrictions – testing if some mapping is a solution is harder than testing if it is a *maximal* solution [23, 2]. By the same token, it was shown in [4] that strictly more severe restrictions on the wdPTs are needed to achieve tractability of the evaluation problem than for the max-evaluation problem. Our complexity analysis of the enumeration problem reveals an interesting effect: it turns out that enumerating the maximal solutions is harder than enumerating all solutions. A yet more detailed picture of the complexity of the enumeration problem (for all solutions resp. for the maximal solutions) is provided by studying also the parameterized complexity of this problem under various restrictions.
- *Data complexity.* In Section 5, we study the data complexity of the enumeration problem (for all resp. for the maximal solutions) of wdPTs. Under the common assumption that matrix multiplication of two Boolean $n \times n$ matrices is not feasible in time $\mathcal{O}(n^2)$, we rule out the possibility of *linear* time preprocessing and constant delay for a very restricted class of wdPTs. However, if we allow *polynomial* time preprocessing then constant delay is achievable for an appropriately restricted class of wdPTs.

2 Preliminaries

Conjunctive queries. We assume familiarity with the relational model, especially with *conjunctive queries (CQs)*, see [1] for details. In the following, we fix some notation. For a set $\mathcal{A}$ of atoms we use $dom(\mathcal{A})$ to denote the set of constants and variables appearing in $\mathcal{A}$, while $var(\mathcal{A})$ refers to the variables only. Similarly, for a mapping μ we denote with $dom(\mu)$ the set of elements on which μ is defined. It is convenient to denote mappings as sets of ordered pairs. Thus two mappings μ_1, μ_2 are equal if $\mu_1 = \mu_2$, and a mapping μ_2 extends a mapping μ_1 if $\mu_1 \subseteq \mu_2$. Furthermore, μ_2 is a proper extension of μ_1 if $\mu_1 \subset \mu_2$.

We write CQs q as $\text{Ans}(\vec{x}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$, where $\vec{x}$ are the *free variables*, and define $var(q) = var(\{R_i(\vec{v}_i) \mid 1 \leq i \leq m\})$. For the *evaluation* $q(\mathcal{D})$ of a CQ q with free variables $\vec{x}$ over database $\mathcal{D}$, we depart slightly from the traditional definition. We define $q(\mathcal{D})$ to be the set of all *mappings* $\mu|_{\vec{x}}$ such that μ is a homomorphism from $R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$ into $\mathcal{D}$, where $\mu|_{\vec{x}}$ denotes the restriction (or projection) of μ to the variables $\vec{x}$ (observe that usually, $q(\mathcal{D})$ is defined as the set of all tuples $\mu(\vec{x})$).

Graphs. We consider undirected, simple graphs $G = (V, E)$. For a graph G, we may write $V(G)$ and $E(G)$ to denote the set of nodes and edges, respectively. As usual, a *tree* is an acyclic graph, and a subtree is a connected subgraph of a tree. A *tree decomposition* of a graph $G = (V, E)$ is a pair (S, ν), where S is a tree and $\nu : V(S) \to 2^V$, that satisfies the following: (1) For each $u \in V$ the set $\{s \in V(S) \mid u \in \nu(s)\}$ is a connected subset of $V(S)$, and (2) each edge of E is contained in one of the sets $\nu(s)$, for $s \in V(S)$. The *width* of (S, ν) is $(\max\{|\nu(s)| \mid s \in V(S)\}) - 1$. The *treewidth* of G is the minimum width of its tree decompositions. Intuitively, the treewidth of G measures its *tree-likeness*. Notice that if G is an undirected graph, then G is acyclic iff it is of treewidth one.

Well-designed pattern trees. *Well-designed pattern trees* (wdPTs) were originally introduced in [23] as a graphical representation of *well-designed SPARQL* defined in [25] and later extended to arbitrary relational vocabulary [4]. Their formal definition is given below.

▶ **Definition 3** (wdPTs). A *well-designed pattern tree* (wdPT) p is a tuple $(T, \lambda, \vec{x})$, such that the following holds:

1. T is a rooted tree and λ maps each node $N \in V(T)$ to a set of relational atoms.
2. For every variable y mentioned in T, the set of nodes of T where y occurs is connected.
3. The tuple $\vec{x}$ of distinct variables from T denotes the *free variables* of the wdPT.

We say that $(T, \lambda, \vec{x})$ is *projection-free*, if $\vec{x}$ contains all variables mentioned in T.

Clearly, CQs correspond to the special case of wdPTs consisting of the root node only. Condition (2) above is referred to as the *well-designedness condition* introduced in [25]. We use upper-case letters to denote nodes of a wdPT, and lower-case letters for vertices of tree-decompositions and general graphs. For a wdPT $p = (T, \lambda, \vec{x})$ and a node $N \in V(T)$, we may abbreviate $var(\lambda(N))$ to $var(N)$. Also, for a subtree T' of T we may use $var(T')$ to denote the set $\bigcup_{N \in V(T')} var(N)$. Finally, we may write $var(p)$ instead of $var(T)$.

Let $p = (T, \lambda, \vec{x})$ be a wdPT. We write R to denote the root of T. Given a subtree T' of T rooted in R, we define $q_{T'}$ to be the CQ $\text{Ans}(\vec{y}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$, where $\{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\} = \bigcup_{N \in T'} \lambda(N)$, and $\vec{y} = var(T')$. Finally, we write $|p|$ to denote the *size* of p in standard relational notation – which corresponds to the size of q_T.

We define the *semantics of wdPTs* by naturally extending their interpretation under semantic web vocabularies [23, 26]. Intuitively, a mapping μ satisfies (T, λ) over a database $\mathcal{D}$ if it is a solution to a CQ $q_{T'}$ and if no proper extension of μ is a solution to some CQ $q_{T''}$. The result of evaluating a wdPT $(T, \lambda, \vec{x})$ over $\mathcal{D}$ contains the projection of all mappings satisfying (T, λ) to $\vec{x}$. We formalize this next.

▶ **Definition 4** (Semantics of wdPTs). Let $p = (T, \lambda, \vec{x})$ be a wdPT and $\mathcal{D}$ a database.

- A *homomorphism* from p to $\mathcal{D}$ is a partial mapping μ for which there is a subtree T' of T rooted in R such that $\mu \in q_{T'}(\mathcal{D})$.
- A homomorphism μ is *maximal* if there is no homomorphism μ' from p to $\mathcal{D}$ s.t. $\mu \subset \mu'$.

The *evaluation* of wdPT $p = (T, \lambda, \vec{x})$ over $\mathcal{D}$, denoted $p(\mathcal{D})$, corresponds to the set of all mappings of the form $\mu|_{\vec{x}}$, such that μ is a maximal homomorphism from p to $\mathcal{D}$.

For a wdPT p and a database $\mathcal{D}$, a mapping μ is called a *partial solution* if there exists some solution $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$. Observe that $p(\mathcal{D})$ may contain mappings μ, μ' s.t. $\mu \subseteq \mu'$. As an example, recall the mappings μ'_1 and μ'_2 that are the result of restricting the mappings μ_1 and μ_2 from Example 2 to the variables y and z. We thus define the set of *maximal solutions* as $p_m(\mathcal{D}) = \{\mu \in p(\mathcal{D}) \mid \text{ for all } \mu' \in p(\mathcal{D})\colon \mu \not\subset \mu'\}$. For instance, in the above example, the maximal solutions contain only the mapping μ'_2.

Parameterized complexity. Let Σ be a finite alphabet. A *parameterization* of Σ^* is a polynomial time computable mapping $\kappa\colon \Sigma^* \to \mathbb{N}$. A *parameterized problem* over Σ is a pair (L, κ) where $L \subseteq \Sigma^*$ and κ is a parameterization of Σ^*. We refer to $x \in \Sigma^*$ as the instances of a problem, and to the numbers $\kappa(x)$ as the parameters. The following well-known problems will play an important role in our parameterized-complexity analyses.

p-Clique	
Instance:	A graph G and $k \in \mathbb{N}$.
Parameter:	k
Question:	Does G contain a clique of size k?

p-Dominating Set	
Instance:	A graph G and $k \in \mathbb{N}$.
Parameter:	k
Question:	Does G contain a dominating set of size k?

A parameterized problem $E = (L, \kappa)$ belongs to the class FPT of "fixed parameter tractable" problems if there exists an algorithm $\mathcal{A}$ deciding L, a polynomial p, and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that the running time of $\mathcal{A}$ is at most $f(\kappa(x)) \cdot p(|x|)$.

Parameterized complexity theory also provides notions of intractability. Towards this notion, we first recall the definition of fpt-reductions. Let $E = (L, \kappa)$ and $E' = (L', \kappa')$ be parameterized problems over the alphabets Σ and Σ', respectively. An *fpt-reduction* from E to E' is a mapping $R\colon \Sigma^* \to (\Sigma')^*$ such that (1) for all $x \in \Sigma^*$ we have $x \in L$ iff $R(x) \in L'$, (2) there is a computable function f and a polynomial p such that $R(x)$ can be computed in time $f(\kappa(x)) \cdot p(|x|)$, and (3) there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

One notion of intractability for parameterized problems are the classes W[i] (for $i \geq 1$) of the W-hierarchy. Since we are here only interested in the classes W[1] and W[2], we omit a discussion of the W-hierarchy and only recall the following facts: A parameterized problem E is in W[1] or W[2], if there exists an fpt-reduction to the problems p-Clique (for W[1]) and p-Dominating Set (for W[2]), respectively. Similarly, E is W[1]-hard or W[2] if there exists an fpt-reduction from p-Clique and p-Dominating Set, respectively. It is strongly believed that problems that are hard for W[1] or W[2] are not in FPT. For details, see [13].

Complexity classes for enumeration problems. A *parameterized enumeration problem* is a triple $(L, \kappa, \mathsf{Sol})$ such that $L \subseteq \Sigma^*$ (for an alphabet Σ), κ is a parameterization of Σ^*, and $\mathsf{Sol}\colon \Sigma^* \to \mathcal{P}(\Sigma^*)$ is a function such that for all $x \in \Sigma^*$, we have that $\mathsf{Sol}(x)$ (the set of "solutions") is finite and $\mathsf{Sol}(x) = \emptyset$ iff $x \notin L$. When we omit the parameterization κ, the pair (L, Sol) is an *enumeration problem*.

An *enumeration algorithm* $\mathcal{A}$ for a (parameterized) enumeration problem $E = (L, \mathsf{Sol})$ (resp., $E = (L, \kappa, \mathsf{Sol})$) is an algorithm which, on input x, outputs exactly the elements from $\mathsf{Sol}(x)$ without duplicates. We denote the output of $\mathcal{A}$ on x by $\mathcal{A}(x)$.

Let $\mathcal{A}$ be an enumeration algorithm for some problem E. For an input x, let $n = |\mathcal{A}(x)|$. For $0 \leq i \leq n$, we define the *delay* $delay(i)$ as follows: $delay(0)$ ("preprocessing") is the time between the start of the algorithm and the (beginning of the) first output (or termination of

$\mathcal{A}$, if $n = 0$). For $0 < i < n$, $delay(i)$ is the time between outputting solution i and $(i+1)$. Finally, $delay(n)$ is the time between the last output and the termination of $\mathcal{A}$.

The concept of the delay between outputs allows us to define several classes of enumeration problems that capture different notions of tractability for these problems [17, 8, 31]. In the sequel, let $E = (L, \mathsf{Sol})$ (resp. $E' = (L, \kappa, \mathsf{Sol})$) be a (parameterized) enumeration problem. For a class $\mathcal{C}$, we say that $E \in \mathcal{C}$ ($E' \in \mathcal{C}$, respectively), if there exists an enumeration algorithm $\mathcal{A}$ for E, some $m \in \mathbb{N}$, and a computable function f such that on every input x the following holds:

- $E \in \mathsf{OutputP}$: $\mathcal{A}$ terminates in time $\mathcal{O}((|x| + |\mathsf{Sol}(x)|)^m)$.
- $E' \in \mathsf{OutputFPT}$: $\mathcal{A}$ terminates in time $\mathcal{O}(f(\kappa(x)) \cdot (|x| + |\mathsf{Sol}(x)|)^m)$.
- $E \in \mathsf{DelayP}$: $delay(i)$ is in $\mathcal{O}(|x|^m)$ for every $0 \leq i \leq |\mathsf{Sol}(x)|$.
- $E' \in \mathsf{DelayFPT}$: $delay(i)$ is in $\mathcal{O}(f(\kappa(x)) \cdot |x|^m)$ for every $0 \leq i \leq |\mathsf{Sol}(x)|$.
- $E \in \mathsf{DelayC_{lin}}$: $delay(0)$ is in $\mathcal{O}(|x|)$ and for $0 < i \leq |\mathsf{Sol}(x)|$, $delay(i)$ is in $\mathcal{O}(1)$.

A stricter notion of allowing for polynomial delay between two solutions is captured by the class $\mathsf{SDelayP}$ ("strict polynomial delay"): An enumeration problem E is in $\mathsf{SDelayP}$ if there exists a total order $<$ on $\mathsf{Sol}(x)$, some $m \in \mathbb{N}$, and an algorithm $\mathcal{B}$ terminating in time $\mathcal{O}(|x|^m)$ with the following output: On input x, it returns the first element of $\mathsf{Sol}(x)$ (according to $<$). On input (x, y) for any $y \in \mathsf{Sol}(x)$, it either returns the next element according to $<$ if there is some, or halts otherwise. As is common in the literature on enumeration, we use RAMs (rather than Turing Machines) as model of computation [17].

3 Parameterized Complexity of Evaluation

While there has been some research on the classical complexity of fragments of wdPTs [4], the parameterized complexity is still open. We thus recall two variants of the evaluation problem, namely Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) (where we ask if a given mapping is a solution or a maximal solution, respectively) and introduce the parameterized versions p-Eval($\mathcal{C}$) and p-Max-Eval($\mathcal{C}$). Here, the size of the query is taken as parameter.

Eval($\mathcal{C}$) / Max-Eval($\mathcal{C}$)	
Instance:	Query: wdPT $p \in \mathcal{C}$, mapping μ, Data: database $\mathcal{D}$.
Question:	$\mu \in p(\mathcal{D})$ / $\mu \in p_m(\mathcal{D})$?

p-Eval($\mathcal{C}$) / p-Max-Eval($\mathcal{C}$)	
Instance:	wdPT $p \in \mathcal{C}$, mapping μ, database $\mathcal{D}$.
Paramter:	$\lvert p \rvert$
Question:	$\mu \in p(\mathcal{D})$ / $\mu \in p_m(\mathcal{D})$?

Observe that in order to talk about the common notions of data- and query complexity, for Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) we distinguish which parts of the input are considered to be part of the query, and which are part of the data. For the parameterized case, because of the explicit parameter, such a distinction does not make sense.

Tractable CQ-evaluation. When looking for tractable classes of wdPT evaluation [4], a natural starting point are tractable classes of CQ evaluation. We define the problem CQ-Eval($\mathcal{C}$) for a class $\mathcal{C}$ of CQs as follows: Given a CQ $q \in \mathcal{C}$, a mapping μ, and a database $\mathcal{D}$ as input, decide if $\mu \in q(\mathcal{D})$. This problem has been extensively studied and several tractable classes of CQs have been identified [32, 14, 15]. If the maximal arity of the relation symbols is known in advance, even the precise border of tractability is known [16].

An important tractable class of CQs is obtained by restricting the *treewidth* of the *Gaifman-graph* of queries [6]. The Gaifman-graph of a CQ q is a graph $G = (V, E)$ where $V = var(q)$ and E contains exactly all pairs of variables that jointly occur in some atom

in q. The treewidth of a CQ is the treewidth of its Gaifman-graph. We denote by $\mathsf{TW}(k)$ the class of CQs of treewidth at most k. It follows from [6] (see also [10]) that for $k \geq 1$, CQ-Eval($\mathsf{TW}(k)$) is in PTIME.

Tractable wdPT-evaluation. In the past, tractable classes of CQs have been successfully used for defining tractable classes of wdPTs following two different approaches. Either by locally restricting the CQs defined by each node of the wdPT to be from some tractable class (*"local tractability"*), or by globally requiring that for every subtree (containing the root) the corresponding CQ is tractable (*"global tractability"*). Formally, let $\mathcal{C}$ be a class of CQs. A wdPT $p = (T, \lambda, \vec{x})$ is *locally in* $\mathcal{C}$ if for every node $N \in V(T)$ the CQ $\text{Ans}() \leftarrow \lambda(N)$ is in $\mathcal{C}$. Also, p is *globally in* $\mathcal{C}$ if for every subtree T' of T rooted in R the CQ $q_{T'}$ is in $\mathcal{C}$. We denote with ℓ-$\mathcal{C}$ and g-$\mathcal{C}$ the sets of all wdPTs that are locally and globally in $\mathcal{C}$, respectively.

It was shown for projection-free wdPTs that local tractability is sufficient to achieve tractability. However, in the presence of projection, Eval(ℓ-$\mathsf{TW}(k)$) and Max-Eval(ℓ-$\mathsf{TW}(k)$) are NP-complete [23] and DP-complete [4], respectively. Resorting to global tractability only helps when dealing with maximal solutions: Max-Eval(g-$\mathsf{TW}(k)$) is in PTIME, while Eval(g-$\mathsf{TW}(k)$) remains NP-complete [4].

Thus, just restricting the structure of the CQs defined by a wdPT is not sufficient for tractability. Instead, an additional source of complexity comes from the information shared between different nodes. Thus another approach is to restrict the number of variables nodes may have in common. Formally, let $p = (T, \lambda, \vec{x})$ be a wdPT. For two nodes $N, M \in V(T)$, we define the interface $\mathcal{I}(N, M) = var(N) \cap var(M)$. Similarly, for a node $N \in V(T)$ define $\mathcal{I}(N) = \bigcup_{M \in (V(T) \setminus \{N\})} \mathcal{I}(N, M)$. Observe that because of condition (2) in the definition of wdPTs, the interface of a node N can be completely determined by just looking at its neighbors, i.e. for every $N \in V(T)$ we have $\bigcup_{M \in (V(T) \setminus \{N\})} \mathcal{I}(N, M) = \bigcup_{\{M \mid \{N,M\} \in E(T)\}} \mathcal{I}(N, M)$. For $c \geq 0$, we say that p has *c-bounded interface* if $|\mathcal{I}(N)| \leq c$ for all $N \in V(T)$. Similarly, we say that p has *c-semi-bounded interface* if for any two distinct nodes $N, M \in V(T)$ we have $|\mathcal{I}(N, M)| \leq c$. We denote with $\mathsf{BI}(c)$ and $\mathsf{SBI}(c)$ the classes of wdPTs of c-bounded interface and c-semi-bounded interface, respectively.

Of course, restricting only the number of shared variables is not sufficient for tractability, since already the evaluation of the CQ in the root node is intractable. However, combining the two approaches above finally leads to tractable classes of wdPTs: Let $\mathcal{C}$ be a class of CQs s.t. CQ-Eval($\mathcal{C}$) is in PTIME and $c \geq 1$. Then Eval(ℓ-$\mathcal{C} \cap \mathsf{BI}(c)$) is also in PTIME [4].

An inspection of the NP-hardness proof for Eval(g-$\mathsf{TW}(k)$) provided in [4] reveals that hardness even holds for Eval(g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$), but this class is newly introduced in the present paper and was not considered in [4]. The introduction of the notion of a semi-bounded interface allows us to define the classes ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ and g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$. Below, we show that a parameterized complexity analysis of these classes helps to explore the gap between tractability of Eval(ℓ-$\mathcal{C} \cap \mathsf{BI}(c)$) and the NP-completeness of Eval(g-$\mathsf{TW}(k)$). In fact, we show that the parameterized evaluation problem for the different classes of wdPTs is in PTIME (for ℓ-$\mathcal{C} \cap \mathsf{BI}(c)$), in FPT (for g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$), W[1]-complete (for ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$), and W[2]-hard (for g-$\mathsf{TW}(k)$), respectively.

Evaluation in case of bounded vs. semi-bounded interface. Before we present our FPT-result for wdPTs in g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$, we first discuss the main difference compared with the restriction to ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$. In the latter case, it is easy to construct a global tree decomposition of width $k + c$, s.t. for any two neighboring nodes N, M in the wdPT, the interface $\mathcal{I}(N, M)$ is covered by some bag in the tree decomposition. Indeed, we can take a

local tree decomposition for every node N (of width at most k) and add all interface variables to every bag. A global tree decomposition of p is then obtained by gluing together the local tree decompositions. With this global tree decomposition, the FPT-membership (actually, even the PTIME-membership) is easily established [4]. In contrast, for semi-bounded interface, the variables in $\mathcal{I}(N, M)$ for neighboring nodes N, M in p can be arbitrarily scattered over the global tree decomposition. Since the total number of interface variables of a single node in p with all its neighbors is unbounded, we cannot apply the above trick to construct a tree decomposition where all interfaces are covered by some bag. Hence, a different approach is needed in the following theorem to obtain FPT-membership also in this case.

We first introduce the decision problem ExtendSolution($\mathcal{C}$) (a similar problem was introduced in [7, 9] to study enumeration problems), defined as follows: given a database $\mathcal{D}$, a wdPT $p \in \mathcal{C}$, a partial mapping μ and a set $\vec{x}' \subseteq \vec{x}$ (where $\vec{x}$ are the free variables of p), does there exist some mapping $\mu' \in p(\mathcal{D})$ such that $\mu \subseteq \mu'$ and $dom(\mu') \cap \vec{x}' = \emptyset$? Observe that the last property $dom(\mu') \cap \vec{x}' = \emptyset$ is essential, since it allows us to explicitly specify variables $\vec{x}'$ which we do not want to be bound by the desired solutions. Clearly, the problem Eval($\mathcal{C}$) corresponds to the special case of the ExtendSolution($\mathcal{C}$) problem where we set $\vec{x}' \cup dom(\mu) = \vec{x}$, i.e., we ask if μ itself is the desired mapping μ'.

▶ **Theorem 5.** *The problem* p-Eval(g-TW(k) ∩ SBI(c)) *is in* FPT *for every* $k, c \geq 1$.

Proof idea. We prove FPT-membership for the ExtendSolution(g-TW(k) ∩ SBI(c)) problem, from which the desired FPT-result follows. Let p be a wdPT and $\mathcal{D}$ a database. Further let $\mathcal{T}$ denote a global tree decomposition of p of width k, let $\vec{x}$ denote the set of free variables in p, and let μ be a mapping with $dom(\mu) \subseteq \vec{x}$. Let $\mathcal{N}$ denote a set of nodes in p with $dom(\mu) \subseteq var(\mathcal{N})$ and let $\mathcal{M} = \{M_1, \ldots, M_\beta\}$ denote the set of nodes outside $\mathcal{N}$ whose parent is in $\mathcal{N}$. We have to test if there exists an extension ν of μ on the existentially quantified variables in $\mathcal{N}$, s.t. ν cannot be further extended to any of the nodes M_i in $\mathcal{M}$.

The key idea is, for all $M_i \in \mathcal{M}$, to define *critical subsets* $\mathcal{C}(N_i, M_i)$ of $\mathcal{I}(N_i, M_i)$, where $N_i \in \mathcal{N}$ is the parent of M_i. Intuitively, $\mathcal{C}(N_i, M_i)$ is defined in such a way that the existence of an extension ν of μ to the variables in M_i only depends on the values for μ on each of the critical subsets. Our FPT-algorithm relies on several crucial properties of $\mathcal{C}(N_i, M_i)$:

First of all, for each critical subset $\vec{v} \subseteq \mathcal{I}(N_i, M_i)$, we can efficiently determine the "good" (resp. "bad") value combinations, i.e., value combinations such that an extension (resp. no extension) of a mapping ν to M_i is possible, namely: $good(\vec{v}) = \{\eta \mid dom(\eta) = \vec{v}$ and there exists an extension ν of η to $var(M_i)$ with $\nu(M_i) \subseteq \mathcal{D}\}$ and $bad(\vec{v}) = \{\eta \mid dom(\eta) = \vec{v}$ and there exists no extension ν of η to $var(M_i)$ with $\nu(M_i) \subseteq \mathcal{D}\}$. It can be shown that, for an arbitrary mapping μ with $\mathcal{I}(M_i, N_i) \subseteq dom(\mu)$, there exists an extension ν of μ with $var(M_i) \subseteq dom(\nu)$ and $\nu(M_i) \subseteq \mathcal{D}$ iff for every $\vec{v} \in \mathcal{C}(M_i, N_i)$, we have $\mu_{|\vec{v}} \in good(\vec{v})$.

Consider μ from our arbitrary instance of ExtendSolution(g-TW(k) ∩ SBI(c)). We have to test if there exists an extension ν of μ to the existentially quantified variables in $\mathcal{N}$, s.t. ν cannot be further extended to any of the nodes M_i in $\mathcal{M}$. In other words, such ν has to satisfy the following two conditions: (1) $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}$ and (2) for every $i \in \{1, \ldots, \beta\}$, there exists a critical subset $\vec{v}_i \in \mathcal{C}(M_i, N_i)$, s.t. $\nu_{|\vec{v}_i} \in bad(\vec{v}_i)$.

Hence, our decision procedure for ExtendSolution(g-TW(k)∩SBI(c)) just checks if such a combination $(\vec{v}_1, \ldots, \vec{v}_\beta)$ of critical subsets exists. We can search for such a combination by nested loops over all $\vec{v}_i \in \mathcal{C}(M_i, N_i)$ with $i \in \{1, \ldots, \beta\}$. Since $\vec{v}_i \subseteq \mathcal{I}(M_i, N_i)$ and $|\mathcal{I}(M_i, N_i)| \leq c$, there are at most 2^c elements in each $\mathcal{C}(M_i, N_i)$. Moreover, β is bounded by the size of p. Hence, we have to check at most $f(p) = (2^c)^{|p|}$ combinations $(\vec{v}_1, \ldots, \vec{v}_\beta)$. To prove the algorithm to be in FPT, it suffices to show that, for a given combination $(\vec{v}_1, \ldots, \vec{v}_\beta)$

of critical subsets, one can test in polynomial time if there exists an extension ν of μ with (1) $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}$ and (2) $\nu_{|\vec{v}_i} \in bad(\vec{v}_i)$ for every $i \in \{1, \ldots, \beta\}$.

The second crucial property of the critical subsets $\mathcal{C}(N_i, M_i)$ with $i \in \{1, \ldots, \beta\}$ is that for any combination $(\vec{v}_1, \ldots, \vec{v}_\beta)$ with $\vec{v}_i \in \mathcal{C}(N_i, M_i)$, we can transform the given global tree decomposition $\mathcal{T}$ of p of width k into a tree decomposition $\mathcal{T}^*$ of width $\leq (k+1) \cdot (c+1)$, s.t. every $\vec{v}_i$ is covered by the bag of some vertex t in $\mathcal{T}^*$. Analogously to the PTIME-membership proof in [4], the tree decomposition $\mathcal{T}^*$ guarantees that the check for the existence of an extension ν of μ with the desired properties (1) and (2) is feasible in polynomial time. ◀

What happens if, instead of the restriction to g-TW$(k) \cap$ SBI(c), we consider ℓ-TW$(k) \cap$ SBI(c)? Below we show that, with this relaxation, fixed-parameter tractability is lost.

▶ **Theorem 6.** *The problem* p-Eval(ℓ-TW$(k) \cap$ SBI(c)) *is* W[1]*-complete for* $k \geq 1$ *and* $c \geq 2$.

Proof sketch. *Membership* is shown by reduction to the W[1]-complete problem Pos-Eval, which is the evaluation problem for FO-queries built from relational atoms using $\exists, \wedge, \vee$ [24] (see also [24] for the problem definition). The idea is to create one positive query which is a big disjunction over all possible CQs $q_{T'}$ corresponding to subtrees of p that potentially have μ as an answer. The maximality of answers is enforced by introducing "interface relations" which, for every child node, only contain those tuples which cannot be extended to the child.

Hardness is shown by an fpt-reduction from p-Clique. Given a graph $G = (V, E)$ and $d \in \mathbb{N}$, we construct a wdPT p and a database $\mathcal{D}$ with the following intuition: The root R of p contains d variables $x_1, \ldots, x_d$. Mapping the root into $\mathcal{D}$ assigns one node from V to each x_i. In addition, R contains one child for each pair of distinct variables $x_\alpha, x_\beta \in \{x_1, \ldots, x_d\}$. Each of these children can be mapped into $\mathcal{D}$ iff there is an edge between the nodes assigned to x_α, x_β. Thus G contains a clique of size d iff there exists a mapping $\mu \in p(\mathcal{D})$ that maps all children of R into $\mathcal{D}$. ◀

We now consider another relaxation of the g-TW$(k) \cap$ SBI(c) restriction from Theorem 5 by considering wdPTs in g-TW(k) but without any bound on the interfaces.

▶ **Theorem 7.** *The problem* p-Eval(g-TW(k)) *is* W[2]*-hard for* $k \geq 1$.

Proof sketch. The proof is by reduction from p-Dominating Set. Thus, let $G = (V, E)$ be a graph and $d \in \mathbb{N}$. The idea of the constructed wdPT p and database $\mathcal{D}$ is to have variables $x_1, \ldots, x_d$ in the root R of p such that a mapping of R into $\mathcal{D}$ assigns one node from V to each x_i. Observe that $S \subseteq V$ is a dominating set iff there does not exist some $u \in V \setminus S$ that is not adjacent to any node in S. This is tested in the single child node of R: It contains an additional variable x_0 which also encodes nodes in V. Now the child node is mapped into $\mathcal{D}$ iff there exists a value for x_0 that differs from those of all x_i's, and there does not exist an edge between the nodes mapped to x_0 and any of the x_i's. I.e., there exists a solution that only maps the root into $\mathcal{D}$ iff G contains a dominating set of size d. ◀

It was shown in [4] that the problem Max-Eval(g-TW(k)) is in PTIME. In other words, for wdPTs with globally bounded treewidth, there is no need to also restrict the interface. Moreover, as recalled above, restricting wdPTs to ℓ-TW$(k) \cap$ BI(c) also yields a restriction of the global treewidth. The only case remaining is therefore the restriction to ℓ-TW$(k) \cap$ SBI(c).

▶ **Proposition 8.** *The problem* p-Max-Eval(ℓ-TW$(k) \cap$ SBI(c)) *is* W[1]*-hard for* $k \geq 1$ *and* $c \geq 2$.

Proof. The reduction used to prove the hardness in the proof of Theorem 6 also proves this case, since clearly $\mu \in p(\mathcal{D})$ iff $\mu \in p_m(\mathcal{D})$. ◀

4 Classical and Parameterized Complexity of Enumeration

We now turn our attention to the enumeration problem. Analogously to the evaluation problem in Section 3, we will study four variants of the enumeration problem, namely enumerating *all* vs. the *maximal* solutions and *parameterized* vs. *non-parameterized* problems.

Enum($\mathcal{C}$) / Max-Enum($\mathcal{C}$)	
Instance:	Query: wdPT $p \in \mathcal{C}$. Data: database $\mathcal{D}$.
Output:	$p(\mathcal{D})$ / $p_m(\mathcal{D})$?

p-Enum($\mathcal{C}$) / p-Max-Enum($\mathcal{C}$)	
Instance:	wdPT $p \in \mathcal{C}$, database $\mathcal{D}$.
Parameter:	$\lvert p\rvert$
Output:	$p(\mathcal{D})$ / $p_m(\mathcal{D})$?

For CQs, the enumeration problem is also well-studied, although not as thoroughly as the evaluation problem. Similarly to the evaluation problem, restrictions on the graph structure of the query have proven useful when looking for tractable enumeration of CQs, cf. [32, 5, 3]. For wdPTs, analogously to the evaluation problem, additional restrictions are necessary in order to achieve tractability. However, for enumeration we get a more diverse picture than for evaluation: When we are interested in all solutions, the additional restrictions used for evaluation are sufficient also for enumeration, while this is not the case if we are only interested in the maximal solutions. It will turn out that the techniques required to analyze the enumeration of *all* vs. the *maximal* solutions differ significantly. We thus treat the enumeration and the max-enumeration problems in separate subsections below.

4.1 Enumeration

For our study of the enumeration problem, the decision problem ExtendSolution($\mathcal{C}$), which we introduced in Section 3, again plays an important role.

▶ **Lemma 9.** *Let $\mathcal{C}$ be a class of pattern trees, $p = (T, \lambda, \vec{x}) \in \mathcal{C}$, and $\mathcal{D}$ a database. Assume that there is a computable function f such that for every partial mapping μ and every subset $\vec{x}'$ of $\vec{x}$, the problem* ExtendSolution($\mathcal{C}$) *can be decided in $\mathcal{O}(f(|p|, |\mathcal{D}|))$. Then there exists an algorithm enumerating $p(\mathcal{D})$ with delay(i) in $\mathcal{O}(f(|p|, |\mathcal{D}|) \cdot |\mathcal{D}| \cdot |p|)$ for $i \geq 0$.*

Proof Sketch. The general idea is to create solutions by iteratively testing if certain variable bindings can be extended to solutions. Observe that for CQs, it would suffice to test (in nested loops) for each variable if it can be bound to a certain domain element. In contrast, for wdPTs, we now have to consider an additional option, namely not binding a variable at all. Thus, we look for extensions (= solutions) that bind some variables to specific domain values (expressed by μ) and leave some variables unbound (expressed by $\vec{x}'$).

In a little bit more detail, we enumerate all $\nu \in p(\mathcal{D})$ as follows: For all $a \in dom(\mathcal{D})$, we can check using ExtendSolution($\mathcal{C}$) whether $\{(x_1, a)\}$ can be extended to a solution (by setting $\mu = \{(x_1, a)\}$), and whether there is a solution $\nu \in p(\mathcal{D})$ with $x_1 \notin dom(\nu)$ (by setting $\vec{x}' = \{x_1\}$). Then by either fixing such a value $a_1 \in dom(\mathcal{D})$, or by fixing the fact that x_1 will not be in the domain of an extension, we can repeat this test for all $a_2 \in dom(\mathcal{D})$. Thus by iteratively fixing such variable assignments (respectively intentional non-assignments) for all n variables x_i in $\vec{x}$ by either extending μ or $\vec{x}'$, we can output a mapping $\rho \in p(\mathcal{D})$ in the n-th step. Given a solution $\rho \in p(\mathcal{D})$, we find the next one by taking the maximal $j \in \{1, \ldots, n\}$ such that under the same assignments for $x_1, \ldots, x_j$, the check for an extension is positive for a different $\nu(x_j)$. Extending this assignment as described above gives a $\rho' \neq \rho$ with $\rho' \in p(\mathcal{D})$. Iterating this process outputs $p(\mathcal{D})$ with a delay of $\mathcal{O}(f(|p|, |\mathcal{D}|) \cdot |\mathcal{D}| \cdot |p|)$. ◀

To make use of this lemma, we identify classes of wdPTs that meet the requirements:

▶ **Proposition 10.** *The following complexity results hold for* ExtendSolution($\mathcal{C}$)*:*
1. *Let $k, c \geq 1$ and $\mathcal{C}$ be a class of CQs for which* CQ-Eval($\mathcal{C}$) *is in* PTIME*. Then* ExtendSolution(ℓ-$\mathcal{C} \cap$ BI(c)) *is in* PTIME*.*
2. ExtendSolution(g-TW(k)) *is* NP*-complete for every $k \geq 1$.*
3. *Let $k, c \geq 1$. Then* ExtendSolution(g-TW(k) $\cap$ SBI(c)) *parameterized by $|p|$ is in* FPT*.*

Proof idea. (1) follows by some slight modifications of the proof of [4, Theorem 6]. Since Eval($\mathcal{C}$) is a special case of ExtendSolution($\mathcal{C}$) where $\vec{x}' = \vec{x} \setminus dom(\mu)$, (2) follows immediately from [4]. For (3) we note that in the previous section, Theorem 5 was stated for p-Eval(g-TW(k) $\cap$ SBI(c)). One can actually show the stronger result of FPT membership for the parameterized version of ExtendSolution(g-TW(k) $\cap$ SBI(c)). ◀

Now the following corollary follows immediately from the previous results.

▶ **Corollary 11.** *Let $k, c \geq 1$.*
- *Let $\mathcal{C}$ be a class of CQs for which* CQ-Eval($\mathcal{C}$) *is in* PTIME*. Then* Enum(ℓ-$\mathcal{C} \cap$ BI(c)) *is in* SDelayP*.*
- *The problem p-*Enum(g-TW(k) $\cap$ SBI(c)) *is in* DelayFPT*.*

We now move to negative results for the enumeration problem. As an important tool and an interesting result in its own right, we establish a close relationship between enumeration and the parameterized complexity of evaluation. We introduce some terminology first:

▶ **Definition 12.** A class $\mathcal{C}$ of wdPTs is *robust* if for every wdPT $p = (T, \lambda, \vec{x}) \in \mathcal{C}$ the following two conditions hold:
1. For every $\mathcal{N} = \{N_1, \ldots, N_m\} \subseteq V(T)$ the wdPT $(T, \lambda_{\mathcal{N}}, \vec{z})$ is in $\mathcal{C}$, where $\vec{z} = \{z_1 \ldots, z_m\}$ is a set of new variables, $\lambda_{\mathcal{N}}(N) = \lambda(N)$ for all $N \in V(T) \setminus \mathcal{N}$ and $\lambda_{\mathcal{N}}(N_i) = \lambda(N_i) \cup \{b(z_i)\}$ for $1 \leq i \leq m$ and some new relation symbol b.
2. For every variable $x \in var(p)$, let p' be the wdPT retrieved from p by replacing every occurrence of x by the same constant c. Then $p' \in \mathcal{C}$.

The notion of "robust" classes of wdPTs is important in the following theorem, to make sure that wdPTs do not fall out of their class when certain transformations are performed.

▶ **Theorem 13.** *Let $\mathcal{C}$ be a robust class of wdPTs. If p-*Enum($\mathcal{C}$) *is in* OutputFPT*, then p-*Eval($\mathcal{C}$) *is in* FPT*.*

Proof Sketch. Given a pattern tree $p = (T, \lambda, \vec{x})$, a database $\mathcal{D}$ and a mapping μ, we first transform p into a pattern tree p' by substituting free variables in p according to μ, and then adding unary atoms of the form $b(z_i)$ with new variables to the leaf nodes of the minimal subtree of T only containing the variables $dom(\mu)$ and to the children of the leaf nodes of the maximal subtree containing only $dom(\mu)$. Further we set the free variables of p' to the newly introduced variables and extend the database to a database $\mathcal{D}' = \mathcal{D} \cup \{b(1)\}$. We can fix a mapping $\mu' \in p'(\mathcal{D}')$ such that $\mu \in p(\mathcal{D})$ iff $\mu' \in p'(\mathcal{D}')$. Since $|p'(\mathcal{D}')|$ is in $\mathcal{O}(|p| \cdot 2^{|p|})$, we can output all of $p'(\mathcal{D}')$ in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^m)$ for some $m \geq 1$ and some function f, and hence decide whether $\mu' \in p'(\mathcal{D}')$ and thus whether $\mu \in p(\mathcal{D})$ in time in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^m)$. ◀

Exploiting this relationship between the evaluation and the enumeration problem, the next results are immediate consequences of the W[1]- and W[2]-hardness, respectively, shown in the previous section and the fact that all the classes mentioned in this paper are robust.

▶ **Corollary 14.** *If* FPT $\neq$ W[1]*, then the following holds:*

- *The problem p-*Enum(ℓ-TW(k) $\cap$ SBI(c)) *is not in* OutputFPT *for* $k \geq 1$ *and* $c \geq 2$*. Thus also the problem* Enum(ℓ-TW(k) $\cap$ SBI(c)) *is not in* OutputP *for* $k \geq 1$ *and* $c \geq 2$*.*
- *The problem p-*Enum(g-TW(k)) *is not in* OutputFPT *for* $k \geq 1$*.*

It follows of course also immediately from Theorem 13 (in combination with Theorem 7) that Enum(g-TW(k)) is not in OutputP if FPT $\neq$ W[1]. However, we can show the same result under an even stronger assumption, namely assuming that PTIME $\neq$ NP, by making use of the following NP-hardness result.

▶ **Proposition 15.** *The following problem is* NP*-hard for every* $k \geq 1$*: Given a wdPT* $p \in g$-TW(k) *and a database* $\mathcal{D}$*, decide whether* $|p(\mathcal{D})| = 2$*.*

Proof sketch. The proof is by reduction from the NP-complete problem Dominating Set, via the same reduction used to prove Theorem 7. W.l.o.g. we consider only graphs G that contain at least one node that is not already a dominating set. Then there always exists one solution that maps both, the root and the child into $\mathcal{D}$. In addition there still exists a second solution that maps only the root into $\mathcal{D}$ iff G contains a dominating set of size d. ◀

The next result follows immediately. Indeed, an algorithm solving Enum(g-TW(k)) in polynomial time w.r.t. the size of the input plus the output would provide a polynomial time decision procedure for Dominating Set, a problem well-known to be NP-hard.

▶ **Corollary 16.** *If* PTIME $\neq$ NP*, then* Enum(g-TW(k)) *is not in* OutputP *for every* $k \geq 1$*.*

4.2 Max-Enumeration

We next turn our attention to the problem of enumerating only the *maximal solutions* of a wdPT. In fact, we show that for none of the classes of wdPTs considered in this work, the problem Max-Enum(ℓ-TW(k) $\cap$ BI(c)) is in OutputP. After establishing this result, we therefore turn towards the parameterized problem, where for all but one of the classes we can show a positive result, namely DelayFPT membership.

Of course, for the negative result on the non-parameterized problem, it suffices to show intractability for Max-Enum(ℓ-TW(k) $\cap$ BI(c)), since the other results follow from that. Again, we do this via an intractability result for a suitable decision problem.

▶ **Proposition 17.** *The following problem is* NP*-hard for every* $k, c \geq 1$*: Given a wdPT* $p \in \ell$-TW(k) $\cap$ BI(c)*, a database* $\mathcal{D}$*, and an integer* $s \geq 1$ *encoded in unary, decide if* $|p_m(\mathcal{D})| > s$*.*

To show that Max-Enum(ℓ-TW(k) $\cap$ BI(c)) is not in OutputP using this result, we recall a relationship from [31] (also [18]). The lemma below is actually a slight reformulation of [31, Lemma 2.11]. However, it holds by the same arguments as the original statement.

▶ **Lemma 18** ([31])**.** *Let* E *be an enumeration problem. If* E *is in* OutputP*, then the following problem is in* PTIME*: Given an instance* x *of* E *and an integer* s *encoded in unary, decide if* $|E(x)| > s$*.*

The intractability of Max-Enum(ℓ-TW(k) $\cap$ BI(c)) now is an immediate consequence of the previous two results.

▶ **Corollary 19.** *If* PTIME $\neq$ NP*, then* Max-Enum(ℓ-TW(k) $\cap$ BI(c)) *is not in* OutputP *for every* $k, c \geq 1$*.*

We now show that the parameterized problem p-Max-Enum($\mathcal{C}$) is tractable in all but one cases. We first establish the tractability for $\mathcal{C} = g\text{-}\mathsf{TW}(k)$ and $\mathcal{C} = \ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c)$, respectively. Of course, this immediately implies tractability for $\mathcal{C} = g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$.

We start by formulating a crucial lemma. It will be convenient to recall the problem Partial-Eval($\mathcal{C}$) from the literature (cf. [4]): Given a wdPT p, a database $\mathcal{D}$ and a mapping μ, does there exist some $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$?

▶ **Lemma 20.** *Let $\mathcal{C}$ be a class of wdPTs such that* Partial-Eval($\mathcal{C}$) *is in* PTIME. *Assume that there exists an enumeration algorithm $\mathcal{A}$, such that for every $p \in \mathcal{C}$ and every database $\mathcal{D}$, $\mathcal{A}$ enumerates some set P of partial solutions with $p_m(\mathcal{D}) \subseteq P$ with delay(i) in $\mathcal{O}(f(|p|)\cdot|\mathcal{D}|^m)$ for some computable function f, some positive integer m and $0 \leq i \leq |P|$. Then p-Max-Enum($\mathcal{C}$) is in* DelayFPT.

Proof idea. Given the enumeration algorithm $\mathcal{A}$ for P, the idea is to construct an extension $\mathcal{A}'$ of $\mathcal{A}$ that, after initializing a set $M = \emptyset$, performs the following steps: (1) Retrieve the next output μ of $\mathcal{A}$ and extend it to a maximal solution μ_m. If μ_m has not been created before, add μ_m to M; (2) If the previous step was repeated $2^{|p|}$ times since the last output, output and delete a mapping from M. If $\mathcal{A}$ halts, $\mathcal{A}'$ outputs M and halts as well.

Two main observations are important: First, if Partial-Eval($\mathcal{C}$) is in PTIME, we can always extend a (partial) solution to a maximal solution by greedily adding variable mappings. Second, at most $2^{|p|}$ partial solutions can be extended to the same maximal solution. ◀

For a class $\mathcal{C}$ of wdPTs to show that Max-Enum($\mathcal{C}$) is in DelayFPT it thus remains to identify a suitable set of partial solutions. We do this for the classes mentioned before.

▶ **Theorem 21.** *Let $\mathcal{C}'$ be a class of CQs s.t.* CQ-Eval($\mathcal{C}'$) *is in* PTIME, *$k, c \geq 1$, and $\mathcal{C} \in \{g\text{-}\mathcal{C}', l\text{-}\mathcal{C}' \cap \mathsf{BI}(c)\}$. Then the problem p-Max-Enum($\mathcal{C}$) is in* DelayFPT.

Proof idea. If CQ-Eval($\mathcal{C}'$) is in PTIME, then the corresponding enumeration problem is in DelayP (cf. [5]). Thus, for wdPTs $p \in g\text{-}\mathcal{C}'$, we have $q_{T'} \in \mathcal{C}'$ for every subtree T' of T containing the root. Thus $\bigcup_{T'} q_{T'}(\mathcal{D})$ gives the required set of partial solutions – it suffices to compute the solutions for one CQ after the other. For $l\text{-}\mathcal{C}' \cap \mathsf{BI}(c)$, we use $p(\mathcal{D})$ which can be efficiently enumerated by Corollary 11. ◀

Algorithm $\mathcal{A}'$ sketched in the proof of Lemma 20 crucially depends on the choice of RAMs as the model of computation: $\mathcal{A}'$ may need to store an exponential number of maximal solutions. A Turing Machine (TM) cannot access these solutions efficiently, while a RAM can. However, these algorithms could be easily adapted to run in *incremental delay* on a TM, i.e. for some $m \in \mathbb{N}$ and computable function f, *delay*(i) is in $\mathcal{O}(f(|p|) \cdot (|p| + |\mathcal{D}| + \sum_{j=1}^{i} |y_i|)^m)$ for $i \geq 0$ (where $y_1, \ldots, y_i$ are the first i solutions returned by the algorithm).

We have thus shown fixed-parameter tractability for three out of the four classes we consider. We conclude this section by showing that for the remaining class the problem is not in OutputFPT.

▶ **Proposition 22.** *If* FPT $\neq$ W[1], *then p-*Max-Enum($\ell\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$) *is not in* OutputFPT *for every $k, c \geq 1$.*

Proof idea. The proof is based on the same parameterized reduction from p-Clique used in the proof of Theorem 6. There we have that $|p_m(\mathcal{D})| \leq 2^{|p|}$. Thus, if enumerating $p_m(\mathcal{D})$ were in OutputFPT, then deciding p-Clique would be in FPT. ◀

5 Constant Delay

We now move to the data complexity of the enumeration problem. To make this explicit in the notation used below, we write ENUM_p to denote the problem $\text{ENUM}(\mathcal{C})$ where the query is considered fixed and the input consists of the database only. The usual goal in the database context is to devise an enumeration algorithm that works with linear time preprocessing and then outputs the solutions with constant delay [29, 30]. To reach this goal, typically additional restrictions on the query (and/or the data) are required than just acyclicity/bounded treewidth [12, 19, 3]. For instance, for CQs it was shown that under reasonable complexity assumptions, the class of conjunctive queries that are *free-connex acyclic* (respectively of *bounded free-connex treewidth*) is the only class of acyclic (bounded treewidth) CQs which allows constant delay enumeration [3].

Below, we show that this goal is not reachable for the classes of wdPTs studied here. Recall from Section 4 that lower bounds are proved relative to some common complexity theoretic assumption such as $\mathsf{PTIME} \neq \mathsf{NP}$ or $\mathsf{FPT} \neq \mathsf{W}[1]$. The assumptions used in this section are of a different nature, namely assuming that certain upper bounds are not achievable for the search of a triangle in a graph [21] and for the multiplication of Boolean matrices [22]. Observe that these are typical problems for showing these kind of lower bounds and have been used e.g. in [3] to show the dichotomy result mentioned above.

We first revisit the case of wdPTs without projections, i.e. let $\mathcal{C}_{\mathsf{pf}}$ be the class of wdPTs $p = (T, \lambda, \vec{x})$ such that $\vec{x} = var(p)$.

▶ **Proposition 23.** *If it is not possible to decide in time $\mathcal{O}(n^2)$ whether a graph G with $|V(G)| = n$ contains a triangle, then ENUM_p is not in $\mathsf{DelayC_{lin}}$ already for wdPTs in $\mathcal{C}_{\mathsf{pf}}$ consisting of only a root node with a single child where the root node contains a single binary atom, the child node contains two binary atoms, and the two nodes share two variables.*

Proof Idea. Constructing a pattern tree p with only two nodes R and N and three binary atoms and an appropriate database $\mathcal{D}$, we can decide whether a graph has a triangle by checking whether there is a $\mu \in p(\mathcal{D})$ such that μ is a mapping on the whole pattern tree. Using the fact that $|p|$ is constant, constant delay enumeration with linear preprocessing thus leads to an algorithm detecting a triangle in $\mathcal{O}(n^2)$. ◀

The negative result in Proposition 23 is mainly due to the restriction of preprocessing to linear time. Below, we show that if we relax this restriction and allow preprocessing in time $\mathcal{O}(|D|^m f(|p|))$ in terms of combined complexity for a constant $m > 0$ and some computable function f, then constant delay is achievable. Note that it is important to require that m be a constant in order to exclude preprocessing of time $\mathcal{O}(|\mathcal{D}|^{|p|})$, which in most cases would suffice to simply compute all of the solutions.

▶ **Theorem 24.** *Let $c, k \geq 1$ be positive integers. Then there exists an enumeration algorithm $\mathcal{A}$ for $\text{ENUM}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c))$ with delay(0) in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^{c+k+1})$ and delay(i) in $\mathcal{O}(1)$ for $i > 0$.*

Proof Idea. In the preprocessing, we construct a global tree decomposition of all atoms, which is consistent with the structure of the pattern tree (cf. our discussion preceding Theorem 5). Then by partitioning the corresponding relations of the nodes of the decomposition and eliminating tuples which are not part of some solution, a repeated top-down traversal through the tree yields all solutions with a delay only in the size of the pattern tree. ◀

Table 1 Summary of the main results on evaluation and enumeration of wdPTs.

	ℓ-TW(k) $\cap$ BI(c)	g-TW(k) $\cap$ SBI(c)	ℓ-TW(k) $\cap$ SBI(c)	g-TW(k)
p-Eval($\mathcal{C}$)	PTIME [4]	FPT	W[1]-complete	W[2]-hard
p-Max-Eval($\mathcal{C}$)	PTIME [4]	PTIME [4]	W[1]-hard	PTIME [4]
Enum($\mathcal{C}$)	SDelayP	*open*	not OutputP	not OutputP
Max-Enum($\mathcal{C}$)	not OutputP	not OutputP	not OutputP	not OutputP
p-Enum($\mathcal{C}$)	SDelayP	DelayFPT	not OutputFPT	not OutputFPT
p-Max-Enum($\mathcal{C}$)	DelayFPT	DelayFPT	not OutputFPT	DelayFPT

We note that the proof in fact allows to show that in the above theorem, $f(|p|)$ is actually polynomial w.r.t. $|p|$. Thus, given an exponential number of solutions, it is impossible to compute all of them during the preprocessing step.

For wdPTs with projection, we need to restrict the class of CQs in the pattern tree for a chance to achieve constant delay with linear preprocessing: Recall that – under reasonable complexity assumptions – the class of free-connex acyclic (respectively of bounded free-connex treewidth) CQs is the only class of acyclic (bounded treewidth) CQs which allows constant delay enumeration [3]. However, we show below that Enum_p is not in $\mathsf{DelayC_{lin}}$ even for wdPTs with such a restriction imposed locally on each set of atoms. As in Proposition 23, constant delay and linear preprocessing would lead to an unlikely upper bound on the runtime of a well-studied combinatorial problem.

▶ **Proposition 25.** *If the product AB of two Boolean $n \times n$ matrices A and B cannot be computed in time $\mathcal{O}(n^2)$, then Enum_p is not in $\mathsf{DelayC_{lin}}$ already for wdPTs consisting of only two nodes, each containing a single binary atom and sharing a single variable.*

Let $\mathcal{C}^*$ be the class of conjunctive queries with bounded free-connex treewidth. Then, even if we allow polynomial time preprocessing instead of a linear one as in Theorem 24, there is no constant delay enumeration algorithm for pattern trees in $\{g\text{-}\mathcal{C}^*, l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)\}$ assuming that $\mathsf{W[1]} \neq \mathsf{FPT}$. This is due to the fact that enumeration in these classes is not in $\mathsf{OutputFPT}$ under this complexity assumption.

▶ **Proposition 26.** *Let $c \geq 1$ and $\mathcal{C}^*$ be the class of conjunctive queries with bounded free-connex treewidth. Further let $p \in \{g\text{-}\mathcal{C}^*, l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)\}$ and $\mathcal{D}$ be a database. Then there is no positive integer m and computable function f such that $p(\mathcal{D})$ can be enumerated with delay(0) in $\mathcal{O}(|D|^m \cdot f(|p|))$ and delay(i) in $\mathcal{O}(1)$ for $i \geq 1$ unless $\mathsf{FPT} = \mathsf{W[1]}$.*

6 Conclusion

In this paper, we have embarked on a complexity analysis of two versions of the enumeration problem of wdPTs: the problems of enumerating all solutions and of enumerating the maximal solutions. Due to the close relationship with the parameterized complexity of the corresponding evaluation problem, we have also revisited the evaluation and max-evaluation problems. A summary of the main results is given in Table 1.

For the problems Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) we have identified fixed-parameter tractable and intractable cases. Likewise, for the two variants of the enumeration problem, we have identified tractable and intractable cases – both in terms of classical and parameterized complexity. More precisely, for the classical complexity, we have established tractability

by showing a strong form of tractability (i.e., polynomial delay) and we have established intractability by ruling out even a weaker form of tractability (i.e., output polynomial time). We have proved analogous results from a parameterized complexity point of view.

Even though we have provided quite a comprehensive picture of the complexities in various settings, Table 1 still calls for further work. Above all, the ENUM($\mathcal{C}$) problem with $\mathcal{C} = g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ is open. We conjecture tractability in this case – but this has to be proved yet. Also, for two of our W[1]- and W[2]-hardness results, a matching upper bound is missing. Finally, the search for tractable classes both, for evaluation and enumeration of wdPTs should be continued. Note that none of the restrictions studied here sufficed to ensure tractability of MAX-ENUM($\mathcal{C}$). Hence, further restrictions should be studied.

Acknowledgments. This work was supported by the Vienna Science and Technology Fund (WWTF) through project ICT12-015 and by the Austrian Science Fund (FWF):P25207-N23. Markus Kröll was funded by FWF project W1255-N23.

References

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

2 Shqiponja Ahmetaj, Wolfgang Fischl, Reinhard Pichler, Mantas Simkus, and Sebastian Skritek. Towards reconciling SPARQL and certain answers. In *Proc. WWW 2015*, pages 23–33. ACM, 2015.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic*, pages 208–222. Springer, 2007.

4 Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. Efficient evaluation and approximation of well-designed pattern trees. In *Proc. PODS 2015*, pages 131–144. ACM, 2015.

5 Andrei A. Bulatov, Víctor Dalmau, Martin Grohe, and Dániel Marx. Enumerating homomorphisms. *J. Comput. Syst. Sci.*, 78(2):638–650, 2012.

6 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

7 Nadia Creignou and J-J Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique théorique et applications*, 31(6):499–511, 1997.

8 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *CoRR*, abs/1306.2171, 2013.

9 Nadia Creignou and Heribert Vollmer. Parameterized complexity of weighted satisfiability problems: Decision, enumeration, counting. *Fundam. Inform.*, 136(4):297–316, 2015. doi:10.3233/FI-2015-1159.

10 Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proc. CP*, pages 310–326, 2002.

11 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, Berlin/Heidelberg/New York, 1999.

12 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proc. PODS 2014*, pages 121–131. ACM, 2014.

13 Jörg Flum and Martin Grohe. Parameterized complexity theory. Berlin/Heidelberg/New York, 2010.

14 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

15 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

16 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), 2007.

17 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

18 Dimitris J Kavvadias, Martha Sideri, and Elias C Stavropoulos. Generating all maximal models of a boolean expression. *Information Processing Letters*, 74(3):157–162, 2000.

19 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proc. PODS 2013*, pages 297–308. ACM, 2013.

20 Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *Proc. ICDT 2015*, volume 31 of *LIPIcs*, pages 212–229. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

21 Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, 407(1):458–473, 2008.

22 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. ISAAC 2014*, pages 296–303. ACM, 2014.

23 Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25, 2013.

24 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

25 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

26 Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In *Proc. PODS 2014*, pages 39–50. ACM, 2014.

27 Reinhard Pichler and Sebastian Skritek. On the hardness of counting the solutions of SPARQL queries. In *Proc. AMW 2014*, volume 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

28 Eric Prud′hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, World Wide Web Consortium (W3C), January 2008. URL: http://www.w3.org/TR/rdf-sparql-query/.

29 Luc Segoufin. Enumerating with constant delay the answers to a query. In *Proc. ICDT 2013*, pages 10–20. ACM, 2013.

30 Luc Segoufin. A glimpse on constant delay enumeration (invited talk). In *Proc. STACS 2014*, volume 25 of *LIPIcs*, pages 13–27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.

31 Yann Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Universite Paris Diderot – Paris 7, December 2010. URL: http://www.prism.uvsq.fr/~ystr/these_strozecki.

32 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB*, pages 82–94, 1981.