On the Hardness of Counting the Solutions of SPARQL Queries*

Reinhard Pichler and Sebastian Skritek

Vienna University of Technology, Faculty of Informatics
{pichler,skritek}@dbai.tuwien.ac.at

1 Introduction

Counting, i.e. deriving the number of answers to a query, is a central feature of every query formalism — take for example the existence of a *count()* aggregate function in basically every major query language.

For SPARQL, the W3C Recommendation for a dedicated query language for RDF data, aggregate functions have been introduced recently [15, 8]. However, despite the ever growing importance of SPARQL, with few exceptions [12, 1, 10], the counting problem for SPARQL queries has remained unexplored so far.

The goal of this work is to initiate a systematic analysis of the complexity of counting the solutions of SPARQL queries. Towards this goal, we start by considering an important core fragment of the language, which extends (Unions of) Conjunctive Queries ((U)CQs) by the *optional* matching feature. This important feature of SPARQL, implemented by the OPT-operator, allows to return *partial answers*. This makes it possible to retrieve meaningful answers even over incomplete data.

Regardless of its importance, counting in the presence of OPT has only been touched in [10]. The remaining research has focused on the SPARQL property paths [1, 12], thus being orthogonal to work on the optionality feature.

(U)CQs on the other hand are the typical starting point for research on query formalisms. For instance, the counting problem #CQ is well-known to be intractable. However, unlike the decision problem it becomes harder when projection is allowed [2]. Recently, there has been renewed interest in #CQ, revealing interesting deviations between tractable instances for the decision and the counting problem in the presence of union and projection [3, 14, 5, 6].

We thus consider the SPARQL fragment of *well-designed graph patterns* [13], extended by projection. This fragment can be considered as an extension of UCQs by the OPT-operator while forbidding certain variable occurrences. Like UCQs in the relational case, well-designed graph patterns not only provide an ideal starting point for research, but constitute an interesting fragment of SPARQL on their own. However, the extension by the OPT-operator imposes new challenges compared to plain (U)CQs. For example, for many fragments containing the OPT-operator, several typical problems studied for query languages become harder (e.g. evaluation, enumeration, testing containment and equivalence). Moreover, even in cases where the complexity remains unchanged, the problems require much more involved algorithms [13, 10].

* This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT12-15 and by the Austrian Science Fund (FWF): P25207-N23.

$\mathcal{S}$	(1)	(2)	(3)	(4)	(5)
$\emptyset$	$\# \cdot$ coNP-c. [10]	#P-c. [10]	#P-c. [10]*	in FP	in FP
$\{\cup\}$	$\# \cdot$ coNP-c.	#P-c.	#P-c.	#P-c.	#P-c.
$\{\pi\}$	$\# \cdot \Sigma_2 P$-c.	$\# \cdot$ NP-c.	$\# \cdot$ NP-c.	#P-c.	#P-c.
$\{\cup, \pi\}$	$\# \cdot \Sigma_2 P$-c.	$\# \cdot$ NP-c.	$\# \cdot$ NP-c.	#P-c.	#P-c.

Table 1. Complexity of #wdSPARQL[$\mathcal{S}$] for (1) arbitrary queries, (2) conjunctions from a tractable fragment of #CQ, (3) conjunctions and complete set of triple patterns from a tractable fragment of #CQ, (4) $\leq k$ variables per conjunction, (5) no AND and complete set of triple patterns from tractable fragment of #CQ; (* implicit in [10])

The **main contribution of this paper** is a detailed complexity analysis of the counting problem for various fragments of well-designed graph patterns. Guided by the experience from #CQ that projection and union have a severe impact on the complexity, we consider the following classes of queries. The core language consists of well-designed graph patterns containing only the AND- and OPT-operator, while UNION and projection are considered as extensions. We use wdSPARQL[$\mathcal{S}$] to denote the class of queries from the core language extended by the features from $\mathcal{S} \subseteq \{\cup, \pi\}$: $\cup$ to denotes UNION and π projection.

Since in the general case, the complexity turns out to be rather high, we look for tractable fragments by considering various restrictions on the structure of the graph patterns (as described in the next section). Our results are summarized in Table 1, with #wdSPARQL[$\mathcal{S}$] denoting the counting problem for a query from wdSPARQL[$\mathcal{S}$], and "c." to abbreviate "completeness". Overall, our results suggest that the count operator makes SPARQL evaluation significantly harder.

2 Problem Definitions and Counting Classes

Following [13], we consider *RDF triples* as tuples in $\mathbf{U}^3$, where $\mathbf{U}$ is an infinite set of URIs. An *RDF graph* is a finite set of RDF triples. The active domain $\text{dom}(G) \subseteq \mathbf{U}$ of an RDF graph G is the set of URIs appearing in G.

Well-Designed Graph Patterns. *Graph patterns (GPs)* form the core of SPARQL. We next formalize (following [13]) the class of well-designed graph patterns (wdGPs) as considered in this paper. Let $\mathbf{V}$ be an infinite set of variables with $\mathbf{U} \cap \mathbf{V} = \emptyset$. A *SPARQL triple pattern (TP)* is a tuple in $(\mathbf{U} \cup \mathbf{V})^3$. A conjunctive pattern is either a TP or a pattern $(P_1 \text{ AND } P_2)$ where P_1, P_2 are conjunctive patterns. An optional pattern is either a conjunctive pattern or a pattern $(P_1 \text{ OPT } P_2)$ where P_1, P_2 are optional patterns. An optional pattern P is *well-designed* if for every subpattern $P' = (P_1 \text{ OPT } P_2)$ of P, every variable that occurs in P_2 and outside P' also occurs in P_1. A *well-designed graph pattern (wdGP)* is either a well-designed optional pattern or a pattern of the form $P_1 \text{ UNION} \ldots \text{UNION } P_n$ where each P_i is a well-designed optional pattern. For a wdGP P we refer to the conjunctive subpatterns of P as "the conjunctions of P". We note that we actually defined the class of wdGP in OPT *normal form* only. However, observe that every wdGP is equivalent to one in OPT normal form [13]. We refer to [13] for further information.

We write vars(P) to denote the set of variables occurring in a GP P. Two variable mappings μ_1 and μ_2 are compatible ($\mu_1 \sim \mu_2$) if they agree on the

shared variables. The *semantics of evaluating a GP P* over an RDF graph G is a set of variable mappings $[\![P]\!]_G$ that is defined recursively as [13]:

1. $[\![t]\!]_G = \{\mu\colon \text{vars}(t) \to \text{dom}(G) \mid \mu(t) \in G\}$ for a triple pattern t.
2. $[\![P_1 \text{ AND } P_2]\!]_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]_G, \mu_2 \in [\![P_2]\!]_G, \text{ and } \mu_1 \sim \mu_2\}$.
3. $[\![P_1 \text{ OPT } P_2]\!]_G = [\![P_1 \text{ AND } P_2]\!]_G \cup \{\mu_1 \in [\![P_1]\!]_G \mid \forall \mu_2 \in [\![P_2]\!]_G\colon \mu_1 \not\sim \mu_2\}$.
4. $[\![P_1 \text{ UNION } \ldots \text{ UNION } P_n]\!]_G = \bigcup_{i=1}^{n} [\![P_i]\!]_G$.

The result of projecting a GP P to a set $\mathbf{X}$ of variables is defined as $[\![(P, \mathbf{X})]\!]_G = \{\mu_{|\mathbf{X}} \mid \mu \in [\![P]\!]_G\}$, where $\mu_{|\mathbf{X}}$ denotes the restriction of μ to the variables in $\mathbf{X}$. We thus consider projection as a result modifier on top of a GP. Finally, note that, as usual [13, 10, 5, 14], we consider set semantics instead of bag semantics.

We can now formalize the classes wdSPARQL[$\mathcal{S}$] of GPs studied in this article (with $\mathcal{S} \subseteq \{\cup, \pi\}$). $\mathcal{S} = \emptyset$ denotes the (well-designed) optional patterns which may be extended by union (if $\cup \in \mathcal{S}$) and/or by projection (if $\pi \in \mathcal{S}$).

Counting Problem. Formally, we study the problem #wdSPARQL[$\mathcal{S}$]:
INSTANCE: $P \in$ wdSPARQL[$\mathcal{S}$], RDF graph G; QUESTION: $|[\![P]\!]_G|$?

Restrictions on GPs. It remains to describe the settings in columns (1)–(5) of Table 1. Case (1) is the general case, i.e. GPs from wdSPARQL[$\mathcal{S}$] without further restrictions. Looking for fragments with a lower complexity, we start by investigating how tractable fragments of #CQ – i.e. *acyclicity* [16] and *quantified star-size* [6] (whose definitions naturally carry over) – behave in our setting.

A CQ (and also a set of triple patterns) is acyclic iff it has a join tree [7]. In the presence of projection however, the existence of a join tree does not guarantee tractability for the counting problem. The idea behind quantified star-size is to divide the atoms in the query into certain components, and to replace each of these components by a new atom only containing free variables. In addition, a new relation for each of these atoms is introduced. The star-size basically denotes the maximal number of "input" relations that need to be joined in order to get these new relations, and thus bounds their size.

Case (2), already described in [10], assumes that counting can be done efficiently for every conjunction P' in the GP. I.e., without projection the set of triple patterns for P' is assumed to be acyclic, while for queries with projection in addition also bounded quantified star-size is assumed. Case (3) assumes in addition that the same properties are satisfied by the set of all TPs occurring in the GP. Since this does not give tractability, case (4) assumes the number of variables in each conjunction to be bounded by some constant (without imposing any additional constraints). This is further strengthened in case (5) where every conjunct must consist of a single TP only and in addition the set of all TPs occurring in the GP is from a tractable fragment of #CQ (like in (3)).

Counting complexity. *Counting problems* can be presented using a *witness* function R which for every input x returns a set $R(x)$ of *witnesses* for x. According to [9], if $\mathcal{C}$ is a complexity class of decision problems, we define $\# \cdot \mathcal{C}$ as the class of all counting problems whose witness function R satisfies the following conditions: (1) there is a polynomial $p(n)$ s.t. for every x and every $y \in R(x)$ we have $|y| \leq p(|x|)$; (2) the problem "given x and y, is $y \in R(x)$?" is in $\mathcal{C}$. For #P – the best known counting complexity class – we have $\#\mathsf{P} = \# \cdot \mathsf{P}$. Moreover, the inclusions $\mathsf{FP} \subseteq \#\mathsf{P} \subseteq \# \cdot \mathsf{NP} \subseteq \# \cdot \mathsf{coNP} \subseteq \# \cdot \Sigma_2 P$ hold [9].

3 Main Results

We next discuss the results presented in Table 1 in more detail. Of course it is not necessary to prove the entry in every cell separately: Clearly, membership carries over from #wdSPARQL[$\mathcal{S}_2$] to #wdSPARQL[$\mathcal{S}_1$] whenever $\mathcal{S}_1 \subseteq \mathcal{S}_2$, while hardness follows along the other direction. Concerning the different cases (1)–(5), in general, hardness carries over to cases with a lower number, while membership also holds for settings with higher numbers. The only exception here is that (2) and (3) are incomparable with (4), i.e. no results carry over between these cases. We note that all hardness results provided in this paper are w.r.t. subtractive reductions, which have been identified as a suitable tool for showing hardness for classes beyond #P [4].

Counting in the general setting. We start with presenting the complexity in the general case (column (1)). For #wdSPARQL[$\emptyset$], the complexity was already classified in [10]. Membership for #wdSPARQL[$\cup$] follows from [13, Corollary 4.9], where the corresponding decision problem was shown to be in coNP.

Corollary 1. #wdSPARQL[$\{\cup\}$] *is* $\# \cdot$ coNP*-complete.*

Analogously to the behaviour of CQs, allowing for projection increases the complexity of the problem. Intuitively, the reason for this is that verifying if some variable mapping is indeed a solution now requires to identify a suitable witness on the existential variables first.

Theorem 1. *The problems* #wdSPARQL[$\{\cup, \pi\}$] *and* #wdSPARQL[$\{\pi\}$] *are* $\# \cdot \Sigma_2 P$*-complete.*

The membership follows from the $\Sigma_2 P$-completeness of the corresponding decision problem, which can be achieved by an extension of [11, Theorem 6.6]. The hardness already holds for GPs containing a single OPT-operator. The main idea of the subtractive reduction (from $\#\Sigma_2$SAT [4]) is to use partial answers to distinguish between satisfying and unsatisfying truth assignments: The first query returns one answer for every possible truth assignment (partial if satisfying, complete otherwise). The other query only returns the complete answers for the unsatisfying assignments. The subtraction thus gives the required solutions.

These results show that counting is highly intractable. Next we thus consider the influence of the restrictions (2)–(5) on the complexity of the problem.

Complexity of the restricted cases. The first presumably easier case, (2), was actually already considered in [10] for #wdSPARQL[$\emptyset$]. There it was shown that although the problem becomes easier, it remains intractable. An inspection of the hardness proof provided in [10] reveals that the constructed GPs already satisfy the additional restriction from case (3), leading to the following result.

Theorem 2 ([10]). *Assume that we only consider GPs where each conjunct is acyclic. Then* #wdSPARQL[$\emptyset$] *is* #P*-complete. It remains* #P*-hard even if in addition the set of all triple patterns occurring in the GP is acyclic.*

A different approach is to restrict the number of variables that are allowed to occur within each conjunct. This allows one to efficiently enumerate all the answers for every conjunct. These local solutions can then be used to compute the overall number of solutions by a traversal of the GP in the style of the Yannakakis algorithm for acyclic CQs [16].

Theorem 3. *#wdSPARQL[∅] is in polynomial time if the number of variables in every conjunction of the graph pattern is bounded by some constant k.*

Adding UNION does not change the complexity of the corresponding decision problem. For the #P-hard cases (2) and (3), the complexity of #wdSPARQL[∅] thus carries over to #wdSPARQL[{∪}]. However, tractability is no longer given for (4) and (5). Intuitively, the hardness stems from the problem of detecting duplicate answers returned by different disjuncts (since we assume set-semantics).

Proposition 1. *Assume that we only consider GPs where every conjunction is acyclic. Then #wdSPARQL[{∪}] is in #P.*

Theorem 4. *Assume that we only consider GPs without* AND *operator and such that the set of all triple patterns occurring in a GP is acyclic. Then #wdSPARQL[{∪}] is #P-hard.*

Also for the settings with projection, all membership results can be derived via the corresponding decision problem. For case (2) of #wdSPARQL[{π}], the NP-membership of the decision problem was already shown in [11, Theorem 6.7].

Proposition 2. *Assume that we only consider GPs where each conjunction is acyclic. Then #wdSPARQL[{∪, π}] is in # · NP.*

The main reason for the #P-membership in the next result is again that the set of answers to each conjunct can be efficiently computed. However, since projection may introduce duplicates, this no longer leads to a tractable counting algorithm.

Theorem 5. *Assume that we only consider GPs where the number of variables within each conjunction is bounded by some constant k. Then #wdSPARQL[{∪, π}] is in #P.*

One of the obstacles for showing hardness (which is required for the cases (3) and (5)) is that the set of all triple patterns occurring in the query has to be acyclic and of bounded quantified star-size. This is achieved by using the optionality feature to simulate an *if/then/else* behaviour by which we make sure that the output variables never share a triple pattern with any other variable. This has the effect that certain variables in the GP are never part of the same answer. This way of circumventing an unbounded star-size is not possible in CQs.

Theorem 6. *Assume that we only consider GPs where both, every conjunction as well as the set of all triple patterns in the GP are acyclic and have a quantified star-size of one. Then #wdSPARQL[{π}] is # · NP-hard.*

Theorem 7. *Assume that we only consider GPs that do not contain the* AND*-operator and where in addition the set of all triple patterns is acyclic and has a quantified star-size of one. Then #wdSPARQL[{π}] is #P-hard.*

Intuitively, the different complexities in these two settings stem from the fact that with a limited number of variables in each conjunct, it is not possible to propagate information arbitrarily throughout the complete query. Instead, every conjunct is restricted to some "local" information.

4 Conclusion

Our study of the counting problem for well-designed graph patterns reveals that counting in the presence of the OPT-operator is significantly harder than for CQs. First, the complexity rises in the general cases. Second, structural parameters that allow for efficient counting for CQs are not sufficient any more: Restrictions based on these properties decrease the complexity, but do not lead to tractability. Third, even quite restrictive assumptions (like forbidding the AND operator) only lead to tractability for queries without UNION and projection.

Our results thus suggest that future work comprising the search for tractable fragments is a very interesting and probably challenging task. One possibility would be to look for suitable adaptations of parameters like quantified star-size to the OPT-operator: The hardness proofs of our results suggest that the current definition of star-size is not suited to deal with all possibilities of partial answers. Other lines of research include restrictions on the nesting structure of OPT.

References

1. M. Arenas, S. Conca, and J. Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *Proc. WWW*, pages 629–638. ACM, 2012.
2. M. Bauland, P. Chapdelaine, N. Creignou, M. Hermann, and H. Vollmer. An algebraic approach to the complexity of generalized conjunctive queries. In *SAT 2004 - Revised Selected Papers*, volume 3542 of *LNCS*, pages 30–45. Springer, 2005.
3. V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theo. Comp. Sci.*, 329(1-3):315–323, 2004.
4. A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theo. Comp. Sci.*, 340(3):496–513, 2005.
5. A. Durand and S. Mengel. Structural tractability of counting of solutions to conjunctive queries. In *Proc. ICDT*, pages 81–92. ACM, 2013.
6. A. Durand and S. Mengel. The complexity of weighted counting for acyclic conjunctive queries. *J. Comput. Syst. Sci.*, 80(1):277–296, 2014.
7. R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.
8. S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Working Draft, W3C, Jan. 2012. http://www.w3.org/TR/sparql11-query/.
9. L. A. Hemaspaandra and H. Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.
10. A. Letelier, J. Pérez, R. Pichler, and S. Skritek. Static analysis and optimization of semantic web queries. In *Proc. PODS*, pages 89–100. ACM, 2012.
11. A. Letelier, J. Pérez, R. Pichler, and S. Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25, 2013.
12. K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *Proc. PODS*, pages 101–112. ACM, 2012.
13. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.
14. R. Pichler and S. Skritek. Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.*, 79(6):984–1001, 2013.
15. E. Prud′hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, W3C, Jan. 2008. http://www.w3.org/TR/rdf-sparql-query/.
16. M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB 1981*, pages 82–94. IEEE Computer Society, 1981.