Towards Faster Reformulation-based Query Answering on RDF Graphs with RDFS Ontologies

Maxime Buron[1][0000−0002−8227−4771], Cheikh-Brahim El Vaigh[2][0000−0002−9843−3001], and François Goasdoué[2][0000−0003−4532−7974]

[1] Univ Oxford, Oxford, United Kingdom
maxime.buron@cs.ox.ac.uk
[2] Univ Rennes, Lannion, France
{cheikh-brahim.el-vaigh,fg}@irisa.fr

Abstract. Answering queries on RDF knowledge bases is a crucial data management task, usually performed through either graph saturation or query reformulation. In this short paper, we optimize our recent state-of-the-art query reformulation technique for RDF graphs with RDFS ontologies [2], and we report on preliminary encouraging experiments showing performance improvement by up to two orders of magnitudes!

Keywords: RDF/S · query answering · reformulation · optimization

1 Introduction

The *RDF* graph data model and its associated *SPARQL* query language are the two well-established W3C standards for sharing data and knowledge, especially on the Semantic Web through DBpedia, Wikidata, or more generally the Linked Open Data cloud. Their rapid adoption over the last decade has made the study of dedicated query answering techniques a hot topic in the data management community at large, to design database management or data integration systems, e.g., [9,3].

Answering queries on an RDF graph (graph in short) is not an easy task as it combines database-style query evaluation with AI-style reasoning, to answer queries from *both* the explicit *and* implicit information that the graph models. *Saturation-based query answering* aims at computing first the *saturation* of the graph by adding to it *all its implicit information* obtained through reasoning. Then, the answers to a query on the graph are obtained by simply posing the query to a data management system (DMS) that stores the graph saturation. This technique provides fast query answering in general, but the saturation needs possibly costly maintenance upon updates [6]. By contrast, *reformulation-based*

query answering amounts to transforming the query into a query *reformulation*, so that the answers to the query are obtained by posing the query reformulation to a DMS that stores only the explicit graph part. This technique does not need maintenance upon updates, as it does not rely on the graph saturation, but query reformulations may be too complex to be efficiently evaluated by a DMS [4].

In this paper, we consider the optimization of the reformulation-based query answering approach that raises so far unsolved performance issues. In particular, we start with our recent technique [2] that applies to the core SPARQL conjunctive queries, a.k.a. *Basic Graph Pattern Queries (BGPQs)*, and RDF graphs with *RDF Schema (RDFS) ontologies*; this technique supports the four RDFS ontological constraints (subclass, subproperty, domain, and range) between classes (i.e., node types) and properties (i.e., edge labels), and all the *RDF entailment rules* [10] that allow reasoning with these constraints. Other techniques only focus on a subset of all these RDF entailment rules [6,4]. Crucially, query answering performance decreases as the number of supported RDF entailment rules increases (because the complexity of its query reformulation increases), hence *efficient* reformulation-based query answering in the setting of [2] is particularly challenging.

Our contributions are twofold. First, in Sec. 2, we identify two causes of limited performance for the reformulation-based query answering technique in [2] and those it encompasses (e.g., [6]), and we propose three optimizations to address them. They all consists in identifying and avoiding useless query reformulation evaluation efforts either by reasoning on a query reformulation itself via *query containment* or by reasoning on a query reformulation based on the queried graph using the *cardinalities* (number of instances) of its classes and properties, or a *summary* of it. Second, in Sec. 3, we discuss in which order these optimizations must be applied to provide best performance and we report on experiments we made to assess the effectiveness of our optimization workflow. As we shall see, our workflow makes reformulation-based query answering (*i*) feasible when it was failing or timing out due to too complex query reformulations, (*ii*) faster when it was feasible (up to 2 orders of magnitude), and (*iii*) better than saturation-based query answering in a non-negligible number of occasions.

2 Optimizing reformulation-based query answering

The reformulation-based query answering technique we devised in [2] computes for a given BGPQ q a reformulation q' of it w.r.t. the RDFS ontology $\mathcal{O}$ of the queried graph $\mathcal{G}$ and the set $\mathcal{R}$ of entailment rules. q' is expressed as a *union of BGPQs (UBGPQ)* that enumerates the – worst case exponentially many – specializations of q w.r.t. $\mathcal{O}$ and $\mathcal{R}$. We present next three optimizations to apply to q' before it is sent for evaluation to some DMS that stores $\mathcal{G}$; they consist in *removing useless BGPQs from* q' to speed up its evaluation time.

Containment-based BGPQ elimination. Because every BGPQ within the UBGPQ reformulation q' is computed *independently* from the others through different reasoning paths, it may happen that q' contains *redundant* BGPQs,

i.e., whose answer sets are *subsets* of those of other BGPQs in q'. These can be identified thus pruned away from q' by comparing every pair of its BGPQs through *containment checking*. We name this first optimization **UBGPQ minimization (M)**. We note that such minimization has been used in the literature in other reformulation-based query answering settings, e.g., existential rules [8].

Cardinality-based and summary-based empty BGPQ elimination. An essential property of reformulation-based query answering, which is common to all reformulation-based techniques and not specific to [2], is that the reformulation q' computed from q w.r.t. $\mathcal{O}$ and $\mathcal{R}$ allows retrieving the answers to q on all the graphs *with ontology* $\mathcal{O}$. Though theoretically nice, the generality of q' needlessly limits its performance: q' may contain *BGPQs with no answer* on the specific queried graph $\mathcal{G}$, while evaluating them may take (significant) time.

BGPQs in q' may have no answer just because $\mathcal{G}$ has no instance for classes or properties these BGPQs mention. To prune them away from q', we compute and store the *cardinalities* of the classes and properties in $\mathcal{G}$, which, clearly, can be maintained very fast upon graph updates. We name this second optimization **empty relation pruning (ER)**.

Also, BGPQs in q' may have no answer because, though every class and property they mention has instances, their selection or join conditions cannot be matched in $\mathcal{G}$. Thus, for each BGPQ in q', we check whether it has no answer on a *summary* of $\mathcal{G}$, in which case it is pruned away from q'. We call this optimization **summary pruning (S)**. To do this, we reuse the notion of RDF graph summary we defined in [5]: a summary is an RDF graph $\mathcal{G}_{\equiv}$ computed as the *quotient graph* of the graph $\mathcal{G}$ to summarize, using an *RDF equivalence relation* $\equiv$ over the nodes of $\mathcal{G}$ (discussed shortly); the particular $\equiv$ to use depends on the target summary usage. We recall that the quotient operation basically fuses every set of equivalent nodes into a single new node representing them all in $\mathcal{G}_{\equiv}$; we say that this latter node *represents* the former equivalent ones, in particular it inherits of all their types and edges. We pointed out in [1] an important property of a summary: a BGPQ q has no answer on $\mathcal{G}$ if the BGPQ $q_{\equiv}$ has no answer on $\mathcal{G}_{\equiv}$, where $q_{\equiv}$ is obtained from q by replacing each mention of a $\mathcal{G}$'s node by the $\mathcal{G}_{\equiv}$'s node it is represented by. We remark here that q may have no answer on $\mathcal{G}$ while $q_{\equiv}$ does have some answer on $\mathcal{G}_{\equiv}$. Importantly, the ability of a summary to prune BGPQs depends on the choice of RDF equivalence relation $\equiv$, all of those presented in [5] have been devised for graph visualization. Therefore, we define a novel RDF equivalence relation for BGPQ pruning that behaves as follows: two nodes $\mathtt{a}$,$\mathtt{b}$ in $\mathcal{G}$ are equivalent, noted $\mathtt{a} \equiv \mathtt{b}$, iff (*i*) they both have the same class (i.e., node type), or (*ii*) they are both equivalent to a third node $\mathtt{c}$ in $\mathcal{G}$, i.e., $\mathtt{a} \equiv \mathtt{c}$ and $\mathtt{b} \equiv \mathtt{c}$. The rationale for this definition is that w.r.t. data compression we want a single representative for every class/type (item (*i*) above) and w.r.t. of the evaluation of BGPQs we want to capture joins between different classes/types (item (*ii*) above).

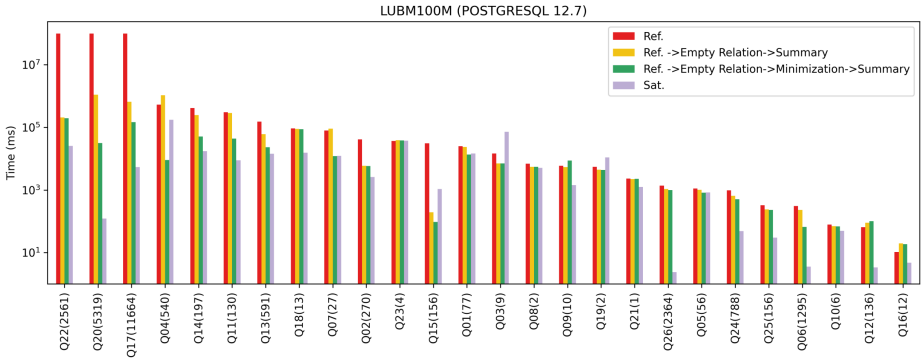

Fig. 1. Query answering times comparison using different optimization workflows.

3 Optimization workflow and experimental results

Optimization workflow. With the above three optimizations in place, the crucial question is now in which order they should be applied to a reformulation q' to maximize the performance gain. After experimenting on the possible orders, the best we found is: ER, then M, and finally S. The intuition behind this optimization workflow is that ER is very fast to perform (linear in the size of q') and may reduce the cost of both M by avoiding containment checks *and* S by avoiding issuing BGPQs to the DMS while we "statically" know that they have no answer. In practice, we observed that ER does help M but not S, because DMSs are smart enough to efficiently detect queries with an empty relation. Then, we chose to apply M because it is *independent* of the size of $\mathcal{G}$ and, though it consists in a number quadratic in the size of q' of NP-Complete BGPQ containment checks, it is performed (i) in-memory and (ii) on BGPQs that are not large in practice (although the UBGPQ q' that unions them may be large). By contrast, the difficulty of S augments with the size of q' and of $\mathcal{G}_{\equiv}$ that grows with the size of $\mathcal{G}$. Applying M before S allows handling larger graphs. In our experiments, the percentage of BGPQs pruned by ER, M and S is respectively in average 32%, 19% and 25%, i.e., 76% in total! Further, while the time spent in applying ER is negligible (always less than 2ms), in average, 3.73% of the query answering time is spent in applying M and 24.97% in applying S (dropping to 14.32%, if we disregard the four queries with no answer for which most of the query answering time is spent in applying S as expected).

Experiments. We have implemented our optimizations in OntoSQL[3] [1,2,4,6], a Java platform for RDF data management on top of Postgres v12.7, in which we stored a LUBM [7] graph $\mathcal{G}$ of 100M edges with its summary $\mathcal{G}_{\equiv}$ of 8.3M edges (7.6% $\mathcal{G}$'s size). For our experiments, we used a CentOs 7.5 linux server with a 2.7GHz Intel Core i7 CPU, 160GB of RAM and a fast HDD. Loading $\mathcal{G}$ took 2h27min and building $\mathcal{G}_{\equiv}$ took 13min. We reused 26 queries from [7,1,4] with 1

[3] `https://ontosql.inria.fr`

to 11 joins and 0 to 20M answers. In Figure 1, we can see their corresponding reformulation size (number of BGPQs in the UBGPQ) *before* optimization (shown as x-axis labels), and their answering times using the reformulation without and with *part of or all* our optimization workflow; all are compared to the saturation-based approach. We set a timeout to 10min; it is reached when answering Q22, Q20 and Q17 without optimization. We observe that our optimization workflow yields performance improvement except on very simple queries where its overhead shows; we remark performance gain up to one order of magnitude for Q20, Q04 and two orders for Q15, and quite surprisingly optimized reformulation-based query answering is faster than saturation-based query answering on Q04, Q15 and Q03.

4 Perspectives

A direct perspective to this work is to support fast summary maintenance upon graph updates. We will investigate this using the *union-find-delete* data structure, which is suited to model and update equivalence relations.

Acknowledgements

This work was partially supported by the ANR project CQFD (ANR-18-CE23-0003).

References

1. Buron, M., Goasdoué, F., Manolescu, I., Merabti, T., Mugnier, M.: Revisiting RDF storage layouts for efficient query answering. In: International Workshop on Scalable Semantic Web Knowledge Base Systems (2020)
2. Buron, M., Goasdoué, F., Manolescu, I., Mugnier, M.: Reformulation-based query answering for RDF graphs with RDFS ontologies. In: ESWC (2019)
3. Buron, M., Goasdoué, F., Manolescu, I., Mugnier, M.: Ontology-based RDF integration of heterogeneous data. In: EDBT (2020)
4. Bursztyn, D., Goasdoué, F., Manolescu, I.: Optimizing reformulation-based query answering in RDF. In: EDBT (2015)
5. Goasdoué, F., Guzewicz, P., Manolescu, I.: RDF graph summarization for first-sight structure discovery. VLDB J. **29**(5) (2020)
6. Goasdoué, F., Manolescu, I., Roatis, A.: Efficient query answering against dynamic RDF databases. In: EDBT (2013)
7. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem. **3**(2-3) (2005)
8. König, M., Leclère, M., Mugnier, M., Thomazo, M.: Sound, complete and minimal ucq-rewriting for existential rules. Semantic Web Journal **6**(5), 451–475 (2015)
9. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: RDFox: A highly-scalable RDF store. In: ISWC (2015)
10. W3C: RDF 1.1 Semantics (2014), `https://www.w3.org/TR/rdf11-mt/`