Heuristics for Applying Cached OBDA Rewritings

Andreas Nakkerud and Evgenij Thorstensen

Department of Informatics, University of Oslo

Abstract. In OBDA systems, cached rewritings can be used to significantly shorten the process of finding rewritings of bigger queries. The main challenge in using cached rewritings is finding the optimal combination to apply, a problem which is NP-complete. In this paper, we present a way to calculate the value of a cached rewriting, and propose using this value to decide which cached rewritings to apply. The idea behind our technique is to estimate how much computation, in particular optimization, has gone into the rewriting. This can tell us something about which cached rewritings we should prioritise, and also when to cache a rewriting. In order to quantify optimization, we define a measure of UCQs, and calculate this measure at different stages of the rewriting.

1 Introduction

Ontology-based data access (OBDA) [11] is a recent paradigm for accessing *data sources* through an *ontology* that acts as a conceptual, integrated view of the data. The data sources are connected to the ontology via *mappings* that specify how to retrieve the appropriate data from the sources. The framework of OBDA has received a lot of attention in the last years: many theoretical studies have paved the way for the construction of OBDA systems (e.g., [3, 5, 13]) and the development of OBDA projects for enterprise data management in various domains [2].

The usual way of answering a query over the ontology in the OBDA framework [11] is to transform it into a set of queries to be executed over the data sources. This process is computationally expensive, as it requires logical reasoning over knowledge represented in the ontology (rewriting) followed by the application of mappings to the ontological query (unfolding). Furthermore, as the resulting set of queries is likely to contain redundancies, optimization techniques are typically applied along the way [10]. It therefore makes sense to *cache* the results of a query rewriting and unfolding, in the hope of using them to speed up the answering of future queries [10]. However, the problem of finding the cached rewritings to use is itself hard, as it is a variant of the well-known problem of query answering using views [8]. In particular, checking whether a cached rewriting can be applied to a query is in general NP-complete.

In this paper, we therefore study the following problem: Given a query Q to answer and a set of cached rewritings of previous queries, how useful is each cached rewriting likely to be, if it is applicable to Q? Knowing this, the system can make sensible decisions as to which cached rewritings to try and apply, and in what order.

As a measure of the usefulness of an existing cached rewriting, previous work by Di Pinto et al. [10] use the number of atoms in the queries. This approach does not take into account the details of the rewriting algorithm used, nor the specification of the OBDA system the queries are answered over. In order to get a finer picture of the usefulness of a cached rewriting, we define a measure for UCQs, and apply it to an intermediate stage of the rewriting. We then use this measure, as well as measures of the cached rewriting, to define a heuristic. In addition to analysing the queries of a cached rewriting directly, this heuristic also takes into account how ontology queries are rewritten by the OBDA system.

2 Preliminaries

In this section, we define basic notions related to OBDA systems and their components: databases, ontologies, and mappings. We then give a definition of query rewriting and unfolding, followed by a formal definition of cached query rewritings as mappings that possess specific properties.

2.1 Databases

In this paper, we assume a fixed relational schema $\mathcal{S}$, and adopt the standard notions for conjunctive queries (CQs) over $\mathcal{S}$ [1]. To every conjunctive query CQ we associate a measure of its size, denoted by $\mathsf{size}(CQ)$. There are several ways to measure the size of a conjunctive query; we will discuss using the number of atoms or the number of variables in the query. However, it is possible to also use more advanced measures of query size, such as tree and hypertree width [6, 7].

Given a conjunctive query Q and a tuple of constants $\boldsymbol{t}$, we write $Q[\boldsymbol{t}]$ for the query obtained by replacing the free variables of Q with the constants in $\boldsymbol{t}$. Given a database instance $\mathcal{D}$ and a query Q, we write $\mathsf{ans}(Q, \mathcal{D})$ for the set of answers to Q over $\mathcal{D}$, defined in the standard way.

2.2 Ontologies

An ontology is a description of a domain of interest in some formal language. Here, we consider the languages of Description Logics (DLs). In general, an ontology expressed in a description logic is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where the TBox $\mathcal{T}$ contains axioms specifying universal properties of the concepts and roles in the domain, while the ABox $\mathcal{A}$ specifies instances of concepts and roles. In the OBDA setting, the ABox is given by mappings rather than explicitly, and so the TBox is the only relevant component. We therefore set $\mathcal{O} = \mathcal{T}$.

In the examples discussed in this paper, we will use the description logic $\mathit{DL\text{-}Lite}_A$ [4, 12]. The syntax of $\mathit{DL\text{-}Lite}_A$ is based on concepts, value-domains, roles, and attributes, and can be defined using the following grammar [10]:

$$\begin{aligned} B &\to A \mid \exists Q \mid \delta(U_C) & E &\to \rho(U) \\ C &\to B \mid \neg B & F &\to T_1 \mid \ldots \mid T_n \\ Q &\to P \mid P^- & V &\to U \mid \neg U \\ R &\to Q \mid \neg Q, & & \end{aligned}$$

where A is a *concept name*, P is a *role name*, U is an *attribute name*, and $T_1, \ldots, T_n$ are *value-domains*. We let Σ_O be a set of ontology predicates, and Γ_C a set of constants. The semantics of *DL-Lite*$_A$ is defined in terms of first-order interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over $\Sigma_O \cup \Gamma_C$. The non-empty domain $\Delta^{\mathcal{I}}$ is the union of the disjoint sets Δ_V and $\Delta_O^{\mathcal{I}}$, where Δ_V is the domain for interpreting data values, and $\Delta_O^{\mathcal{I}}$ is the domain for interpreting object constants. The interpretation function $\cdot^{\mathcal{I}}$ is defined as follows:

$$\begin{aligned} A^{\mathcal{I}} &\subseteq \Delta_O^{\mathcal{I}} & (\neg U)^{\mathcal{I}} &= (\Delta_O^{\mathcal{I}} \times \Delta_V) \setminus U^{\mathcal{I}} \\ P^{\mathcal{I}} &\subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} & (P^-)^{\mathcal{I}} &= \{(o, o') \mid \exists v.[(o', o) \in P^{\mathcal{I}}]\} \\ U^{\mathcal{I}} &\subseteq \Delta_O^{\mathcal{I}} \times \Delta_V & (\exists Q)^{\mathcal{I}} &= \{o \mid \exists o'.[(o, o') \in Q^{\mathcal{I}}]\} \\ (\neg B)^{\mathcal{I}} &= \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}} & (\delta(U))^{\mathcal{I}} &= \{o \mid \exists v.[(o, v) \in U^{\mathcal{I}}]\} \\ (\neg Q)^{\mathcal{I}} &= (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}} & (\rho(U))^{\mathcal{I}} &= \{v \mid \exists o.[(o, v) \in U^{\mathcal{I}}]\}. \end{aligned}$$

A *DL-Lite*$_A$ TBox $\mathcal{T}$ is a finite set of axioms of the form

$$B \sqsubseteq C \qquad Q \sqsubseteq R \qquad E \sqsubseteq F \qquad U \sqsubseteq V \qquad (\mathsf{funct}\ Q) \qquad (\mathsf{funct}\ U).$$

The interpretation $\mathcal{I}$ satisfies an axiom $X \sqsubseteq Y$ if $X^{\mathcal{I}} \subseteq Y^{\mathcal{I}}$. $\mathcal{I}$ satisfies $(\mathsf{funct}\ Z)$ if for every $o, o', o'' \in \Delta_O^{\mathcal{I}}$, if $(o, o') \in Z^{\mathcal{I}}$ and $(o, o'') \in Z^{\mathcal{I}}$, then $o' = o''$.

2.3 Mappings

In the general case, a mapping assertion m between a database schema $\mathcal{S}$ and an ontology $\mathcal{O}$ has the form $Q \rightsquigarrow q$, where the body Q is a CQ over $\mathcal{S}$, and the head q a CQ over the vocabulary of $\mathcal{O}$ [11], possibly with shared free variables. To define the semantics of mapping assertions, we first need to define the notion of an OBDA system.

2.4 OBDA systems

An OBDA system specification is a triple $\mathcal{B} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{S}$ is a database schema, $\mathcal{M}$ a set of mapping assertions, and $\mathcal{O}$ an ontology. The semantics of query answering in OBDA systems are usually defined using first-order interpretations.

Definition 1 (OBDA semantics). *Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system specification, and $\mathcal{D}$ a database for $\mathcal{S}$. A first order interpretation $\mathcal{I}$ with $\mathcal{I} \models \mathcal{D}$ is a model for $\mathcal{B}$ if*

- *$\mathcal{I} \models \mathcal{O}$, and*
- *for every tuple of constants $\boldsymbol{t}$ from $\mathcal{D}$, and every mapping assertion $Q \rightsquigarrow q \in \mathcal{M}$, we have that $\mathcal{I} \models q[\boldsymbol{t}]$ whenever $\mathcal{I} \models Q[\boldsymbol{t}]$.*

The set of answers to a query q over $\mathcal{B}$ and $\mathcal{D}$ is the set of tuples $\boldsymbol{t}$ from $\mathcal{D}$ such that $q[\boldsymbol{t}]$ is true in every model of $\mathcal{B}$ and $\mathcal{D}$. We write $\mathsf{ans}(q, \mathcal{B}, \mathcal{D})$ *for the answers to q over $\mathcal{B}$ and $\mathcal{D}$.*

As mentioned in the introduction, answering a query in an OBDA system is usually done by rewriting and unfolding the query into the correct database queries to execute.

Definition 2 (Rewriting and unfolding). *Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system specification, and q a CQ over the vocabulary of $\mathcal{O}$. A* rewriting *of q under $\mathcal{B}$ is a query q' over the same vocabulary such that* $\mathsf{ans}(q', \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, \mathcal{D}) = \mathsf{ans}(q, \mathcal{B}, \mathcal{D})$ *for every database instance $\mathcal{D}$.*
An unfolding *of the rewriting q' of q is a query Q over $\mathcal{S}$ such that* $\mathsf{ans}(Q, \mathcal{D}) = \mathsf{ans}(q, \mathcal{B}, \mathcal{D})$ *for every database instance $\mathcal{D}$.*

We can now define the notion of a perfect mapping assertion, which captures the idea of caching the rewriting and unfolding of a query.

Definition 3 (Perfect mapping assertion [10]). *Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system specification. A mapping assertion $Q \rightsquigarrow q$ is a* perfect mapping assertion *if for every database instance $\mathcal{D}$, we have* $\mathsf{ans}(q, \mathcal{B}, \mathcal{D}) = \mathsf{ans}(Q, \mathcal{D})$.

Each cached rewriting and unfolding is a perfect mapping assertion. In fact, caching is one of the primary methods for obtaining perfect mapping assertions [10].

3 Applying Perfect Mapping Assertions

As part of their full OBDA rewriting algorithm, Di Pinto et al. [10] present the algorithm *ReplaceSubqueryR* for applying a perfect mapping assertion before the regular rewriting procedure starts. Using the notion of *restricted homomorphisms,* they give an exact definition of when a perfect mapping assertion can be applied.

Given a conjunctive query q, *ReplaceSubqueryR* goes through the perfect mapping assertions in the order specified by some heuristic, modifying q whenever the perfect mapping assertion is applicable. The order of application is significant, since the application of one perfect mapping assertion can prohibit the subsequent application of another. Finding the optimal combination of perfect mapping assertions to apply is NP-hard [10]. Di Pinto et al. use a greedy strategy. The heuristic for this strategy is the number of atoms in the heads of the perfect mapping assertions.

Our goal is to define an improved heuristic for the order of application of perfect mapping assertions.

The following example, based on one of the challenges faced by the Optique project[1], shows how we create perfect mapping assertions.

Example 4. Let $company(name, owner, manager, accountant)$ be a database table. We define the concepts $Company$ and $Owner$, $Manager$ and $Accountant$, and the roles $hasOwner$, $hasManager$, and $hasAccountant$. We define the TBox,

$$\mathcal{T} = \left\{ \begin{array}{r} \exists hasOwner \sqsubseteq Company, \\ \exists hasManager \sqsubseteq Company, \\ \exists hasAccountant \sqsubseteq Company \end{array} \right\},$$

and mapping assertions

$$\mathcal{M} = \left\{ \begin{array}{rl} \exists y, z, w.company(x, y, z, w) & \rightsquigarrow Company(x) \\ \exists z, w.company(x, y, z, w) & \rightsquigarrow hasOwner(x, y) \\ \exists y, w.company(x, y, z, w) & \rightsquigarrow hasManager(x, z) \\ \exists y, z.company(x, y, z, w) & \rightsquigarrow hasAccountant(x, w) \end{array} \right\}.$$

We now look at the query $q(x) = Company(x)$. The ontology rewriting of $Company(x)$ is

$$\begin{aligned} q'(x) = {} & Company(x) \lor \exists v.hasOwner(x, y) \\ & \lor \exists v.hasManager(x, y) \lor \exists v.hasAccountant(x, y). \end{aligned}$$

When unfolding $q'(x)$ we get the UCQ

$$\begin{aligned} & \exists y, z, w.company(x, y, z, w) \\ & \quad \lor \exists v, z, w.company(x, v, z, w) \\ & \quad \lor \exists y, v, w.company(x, y, v, w) \\ & \quad \lor \exists y, z, v.company(x, y, z, v), \end{aligned}$$

which is obviously equivalent to

$$\exists y, z, w.company(x, y, z, w).$$

We cache this rewriting by saving the perfect mapping assertion

$$\exists y, z, w.company(x, y, z, w) \rightsquigarrow Company(x).$$

Example 4 illustrates one situation where perfect mapping assertions are useful. An alternative way of dealing with his particular example is by modifying the set $\mathcal{M}$ of mapping assertions according to the TBox [13], and then to optimise it [9, 13]. This approach is not as general as the perfect mapping assertion approach, because it only works when there are redundancies in the mapping assertions. The perfect mapping assertions can represent optimisations that are only valid in special cases.

[1] http://optique-project.eu/

4 Query Measure

In order to evaluate the quality of a perfect mapping assertion, we need some way of measuring a UCQ. We will use this measure to quantify the amount of optimization that has gone into creating a perfect mapping assertion.

There are several ways to measure the size of a conjunctive query. For ease of exposition, we will use the number of atoms in the query in our examples, although the number of variables is usually more relevant to the exact cost of optimising a query, since query optimisation amounts to finding homomorphisms between queries. If the number of variables is approximately linear in the number of atoms, the results of using either will be similar.

Furthermore, the number of atoms in each conjunction of the unfolding of a conjunctive query can be approximated from the number of mapping assertions that mentions conjunct. On the other hand, finding the number of variables requires an analysis of each mapping assertion.

The size of a UCQ is harder to describe with a single number, because UCQs have two dimensions: the conjunctions and the conjuncts in them.

Definition 5 (Measure of UCQs). *Let* $Q = CQ_1 \vee \ldots \vee CQ_k$ *be a UCQ,* $\mathsf{size}(CQ_i)$ *the size (e.g. number of atoms) of* CQ_i*, and* f *a weighting function. Define* $g(x) = x$ *if* f *is at most linear, and* $g = f^{-1}$ *otherwise. The* measure of Q *is*

$$S_Q = g\left(\sum_{i=1}^{k} f(\mathsf{size}(CQ_i))\right).$$

The function f defines how S_Q depends on the size of the conjunctive queries, the function g makes S_Q at most linear in the sum of the sizes. Our choice of g will make S_Q of the order $O(k \cdot \max_i[\mathsf{size}(CQ_i)])$, where, and k is the number of conjunctive queries in the UCQ. We choose f according to the following observations.

- f **constant:** We disregard the size of disjuncts. The cost of performing optimizations is assumed to be insignificant compared to the cost of rewriting, unfolding and storing the query.
- f **linear:** The cost of optimization is assumed to be on the same order as the cost of rewriting, unfolding and storing the query.
- f **polynomial or exponential:** The cost of optimization is assumed to be more important than the cost of rewriting, unfolding and storing the query.

The above list is explains how different definitions of f affects the measure S_Q. It gives a strong indication of how we should define f, depending on what we want to use the measure S_Q for. In Sections 5 and 6, we will use linear f. We get back to the choice of f briefly in Section 7.

5 Maximal expansion

Having defined a measure for UCQs, we are now in a position to assign a value to the body Q of a perfect mapping assertion $Q \rightsquigarrow q$. Such an analysis gives us some information about how much we stand to gain by applying this perfect mapping assertion. We can, however, get an even better picture by looking at the process of rewriting q. Doing this, we can get a measure of how complex q really is in the relevant OBDA system.

Definition 6 (Maximal expansion). *Let q be a query, $\mathcal{B} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ an OBDA system specification, and $\mathfrak{R}$ an ontology rewriting algorithm. The* maximal expansion of q over $\mathcal{B}$ and $\mathfrak{R}$, *denoted* $\mathsf{me}(q, \mathcal{B}, \mathfrak{R})$, *is the unoptimised ontology rewriting and unfolding of q over $\mathcal{B}$ using $\mathfrak{R}$.*

We write $\mathsf{me}(q, \mathcal{B})$ when the choice of $\mathfrak{R}$ is clear from the context. When the OBDA system is also understood from the context, we let $S_{\mathsf{me}(q)}$ denote the measure of $\mathsf{me}(q, \mathcal{B})$. The following example shows how we calculate $S_{\mathsf{me}(q)}$.

Example 7. We look at an OBDA system specification with TBox axioms $A \sqsubseteq \exists R$ and $S \sqsubseteq T$, and the mapping

$$\mathcal{M} = \left\{ \begin{array}{l} Q_{A_1} \rightsquigarrow A, Q_{A_2} \rightsquigarrow A, Q_{A_3} \rightsquigarrow A, \\ Q_{R_1} \rightsquigarrow R, Q_{R_2} \rightsquigarrow R, Q_{R_3} \rightsquigarrow R, Q_{R_4} \rightsquigarrow R, \\ Q_{S_1} \rightsquigarrow S, Q_{S_2} \rightsquigarrow S, \\ Q_{T_1} \rightsquigarrow T, Q_{T_2} \rightsquigarrow T, Q_{T_3} \rightsquigarrow T \end{array} \right\}$$

and rewrite the query $q(x, y) = \exists z.R(x, z) \wedge T(x, y)$. The ontology rewriting of q according to rewriting algorithm $\mathfrak{R}$ is

$$\begin{aligned} q'(x, y) = {} & [\exists z.R(x, z) \wedge T(x, y)] \vee [\exists z.R(x, z) \wedge S(x, y)] \\ & \vee [A(x) \wedge T(x, y)] \vee [A(x) \wedge S(x, y)] \end{aligned}$$

We choose $f(x) = g(x) = x$, let $\mathsf{size}(CQ)$ be the number of atoms in CQ, and calculate $S_{\mathsf{me}(q)}$. If each Q_{X_n} is a query without joins, then the resulting maximal expansion is a UCQ with joins of size 2. The first disjunct in q' is unfolded into 12 conjunctive queries of size 2, since there are four queries mapped to R and 3 to T. We do similar calculations with each disjunct of q', and end up with a total of 35 conjunctive queries, each of size 2. Therefore $S_{\mathsf{me}(q)} = 70$.

In Example 7, we assumed that there were no conflicts between mapping assertions during the unfolding. Such conflicts can arise when there the mapping assertions contain constants in place of some variables. In this case, only those mapping assertions that have constants replacing the same variables can create conflict, and this will usually only be the case for a very few combinations of assertions. If constants are common in the used mapping assertions, then greater care must be taken when using approximations of $S_{\mathsf{me}(q)}$.

6 Heuristic

With a measure for UCQs and the notion of maximal expansions, we are now in a position to expand on the heuristic suggested by Di Pinto et al. [10]. We design a heuristic where large values are better. In the following we assume a fixed OBDA system and ontology rewriting algorithm.

For a perfect mapping assertion $Q \rightsquigarrow q$, we calculate the measures S_q, S_Q and $S_{\mathsf{me}(q)}$. Note that $S_q = \mathsf{size}(q)$, since q is a conjunctive query.During both ontology rewriting and unfolding, the size of a conjunctive query can grow exponentially. In both cases there is generally multiple options for dealing with every conjunct, and the result is a very large UCQ. For this reason, we will use $\log S_Q$ and $\log S_{\mathsf{me}(q)}$, since they will be approximately proportional to S_q. We also calculate the ratio $\log S_{\mathsf{me}(q)} / \log S_Q$, which tells us how much optimisation has gone into the creation of the perfect mapping assertion. We use this optimization ratio to adjust other measures of the value of the perfect mapping assertion.

If S_q is large for a perfect mapping assertion $Q \rightsquigarrow q$, that is, the size of q is large, then applying it cuts a large portion of the original query. There is, however, a risk that this portion could be rewritten directly without much difficulty. In order to compensate for this effect, we scale S_q by the optimisation ratio $\log S_{\mathsf{me}(q)} / \log S_Q$.

A perfect mapping assertion is also valuable if the corresponding $S_{\mathsf{me}(q)}$ is large, no matter the size of S_q. Again, we scale $\log S_{\mathsf{me}(q)}$ by the optimisation ratio $\log S_{\mathsf{me}(q)} / \log S_Q$.

Assigning a weight parameter to each of these compound measures, we get the heuristic

$$a \frac{(\log S_{\mathsf{me}(q)})^2}{\log S_Q} + b \frac{S_q \cdot \log S_{\mathsf{me}(q)}}{\log S_Q} + c S_q$$

We assume S_q, $\log S_Q$ and $\log S_{\mathsf{me}(q)}$ have the same units, since they are all approximately linear in the size of the head q of the perfect mapping assertion $Q \rightsquigarrow q$. Then, the unit of each term in the above sum is the same, and also approximately linear in the $\mathsf{size}(q)$. This means that the ratio between the terms will be fairly stable in respect to typical query size, and as such, the tuning of the parameters a, b, and c will not be very sensitive to the typical query size.

Example 8. We look at an ontology with an empty TBox, and the four roles R_1, R_2, R_3, and R_4. We define the set of mapping assertions

$$\mathcal{M} = \{Q_i^j \rightsquigarrow R_i \mid j = 1, \ldots, i^2\},$$

where $i = 1, \ldots, 4$. We let $\mathsf{size}(CQ)$ be the number of atoms in CQ, and assume that each Q_i^j is atomic. We let $f(x) = x$, and calculate S_q and $S_{\mathsf{me}(q)}$ for different conjunctions of R_i, with each role occurring at most once in each query. The results are shown in Table 1. We see that $S_{\mathsf{me}(q)}$ gives us a much finer division than S_q, which only divides the queries into 3 groups. Given the query

$$R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge R_3(x_3, x_4) \wedge R_4(x_4, x_5),$$

and the perfect mapping assertions

$$\begin{aligned} Q_A(y_1, y_2, y_3, y_4) &\rightsquigarrow R_1(y_1, y_2) \wedge R_2(y_2, y_3) \wedge R_3(y_3, y_4) \\ Q_B(z_1, z_2, z_3) &\rightsquigarrow R_3(z_1, z_2) \wedge R_4(z_3, z_4), \end{aligned}$$

there is a good chance that our best option is to apply the second, shorter perfect mapping assertion, because the maximal expansion of $R_3(x_3, x_4) \wedge R_4(x_4, x_5)$ is so large. In order to be more precise, we would need to obtain the measures S_{Q_A} and S_{Q_B}.

q	R_1	R_2	R_3	R_4	R_1 R_2	R_1 R_3	R_1 R_4	R_2 R_3	R_2 R_4	R_3 R_4	R_1 R_2 R_3	R_1 R_2 R_4	R_1 R_3 R_4	R_2 R_3 R_4
S_q	1	1	1	1	2	2	2	2	2	2	3	3	3	3
$S_{\mathsf{me}(q)}$	1	4	9	16	8	18	32	72	128	288	108	192	432	1728

Table 1. The measure $S_{\mathsf{me}(q)}$ can be used to fine order the groups defined by S_q, but there are also pairs of queries where S_q and $S_{\mathsf{me}(q)}$ disagree on ordering.

7 Discussion and Conclusion

Di Pinto et al. [10] have found that using cached rewritings can significantly reduce the cost of answering queries. Their well performing algorithm, *ReplaceSubqueryR*, relies on a heuristic for determining the order in which to apply cached rewritings. Di Pinto et al. have chosen a simple heuristic based on the sizes of the heads of the cached rewritings. This is a good choice if all subqueries have approximately the same complexity when seen together with the TBox and the mapping. If some ontology predicates are easier to rewrite and unfold than others, then the measures of the maximal expansion $S_{\mathsf{me}(q)}$ and the optimised rewriting S_Q become relevant to the choice of cached rewriting.

We suggest the heuristic

$$a\frac{(\log S_{\mathsf{me}(q)})^2}{\log S_Q} + b\frac{S_q \cdot \log S_{\mathsf{me}(q)}}{\log S_Q} + cS_q$$

If we let $a = b = 0$, $c \neq 0$, and define $\mathsf{size}(CQ)$ to be the number of atoms in CQ, then our heuristic becomes the same as the one used in [10]. By tweaking f in Definition 5, and the parameters a, b and c, we can shift importance away from S_q, and towards maximal expansion $S_{\mathsf{me}(q)}$ and optimisation S_Q. The ideal setup will depend on the OBDA system specification, and should be decided experimentally. For the heuristic presented here to be more accurate than the one in [10], the OBDA system specification needs to be uneven in terms of how each ontology predicate is rewritten. If the mapping has very many assertions for some ontology predicates and few for others, or if some ontology predicates occur often in the TBox while others don't, then the count of predicates in a query need not reflect how it behaves during rewriting. Also, if the mapping is very large, the optimisation ratios are more likely to be significant, since the unfolding and subsequent optimisation will be a large part of query rewriting.

Since S_q, $S_{\mathsf{me}(q)}$, and S_Q are relatively cheap to calculate or approximate during rewriting, and cheap to store in a cache, we claim our suggested heuristic, in many settings, will outperform the simpler heuristic provided by [10]. Even when the simple heuristic performs very well, we can let $a = 0$ and $b \ll c$, so that $S_{\mathsf{me}(q)}$ is used for a fine splitting as illustrated by Example 8.

8 Future Work

We plan to continue this work by experimentally verifying our results. In particular, we would like to compare the different weighting and scaling functions discussed here on real-world datasets. Another line of enquiry would be to see how well these heuristics can substitute for e.g. techniques to eliminate mapping redundancy, as discussed in Section 3.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. Natalia Antonioli, Francesco Castanò, Cristina Civili, Spartaco Coletta, Stefano Grossi, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Domenico Fabio Savo, and Emanuela Virardi. Ontology-based data access: the experience at the Italian Department of Treasury. volume 1017, pages 9–16, 2013.
3. D. Calvanese, M. Giese, P. Haase, I. Horrocks, T. Hubauer, Y. Ioannidis, E. Jiménez-Ruiz, E. Kharlamov, H. Kllapi, J. Klüwer, M. Koubarakis, S. Lamparter, R. Möller, C. Neuenstadt, T. Nordtveit, Ö. Özcep, M. Rodriguez-Muro, M. Roshchin, F. Savo, M. Schmidt, A. Soylu, A. Waaler, and D. Zheleznyakov. Optique: OBDA solution for big data. In *Revised Selected Papers of ESWC 2013 Satellite Events*, volume 7955 of *LNCS*, pages 293–295. Springer, 2013.
4. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
5. Cristina Civili, Marco Console, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Lorenzo Lepore, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, Valerio Santarelli, and Domenico Fabio Savo. MASTRO STUDIO: Managing ontology-based data access applications. *PVLDB*, 6:1314–1317, 2013.
6. Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

7. Georg Gottlob, Reinhard Pichler, and Fang Wei. Tractable database design through bounded treewidth. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'06)*, pages 124–133. ACM, 2006.
8. Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
9. Domenico Lembo, José Mora, Riccardo Rosati, Domenico Fabio Savo, and Evgenij Thorstensen. Towards mapping analysis in ontology-based data access. In Roman Kontchakov and Marie-Laure Mugnier, editors, *Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, volume 8741 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 2014.
10. Floriana Di Pinto, Domenico Lembo, Maurizio Lenzerini, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Optimizing query rewriting in ontology-based data access. In Giovanna Guerrini and Norman W. Paton, editors, *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 561–572. ACM, 2013.
11. Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. In Stefano Spaccapietra, editor, *Journal on Data Semantics X*, volume 4900 of *Lecture Notes in Computer Science*, pages 133–173. Springer, 2008.
12. Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
13. Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 558–573. Springer, 2013.