Evaluation of Query Rewriting Approaches for OWL 2

Héctor Pérez-Urbina, Edgar Rodríguez-Díaz, Michael Grove,
George Konstantinidis, and Evren Sirin

Clark & Parsia, LLC
United States
{hector,edgar,mike,george,evren}@clarkparsia.com

Abstract. Query answering over ontologies is a crucial feature in contexts such as ontology-based data access and semantic information integration. There is considerable research interest in using *query rewriting* for efficient and scalable query answering: instead of evaluating a given query over the ontology with the (potentially very large) data directly, one rewrites the query with respect to the relevant knowledge in the ontology, and delegates the evaluation of the computed rewriting to a (possibly deductive) database system where the data resides. In this paper we examine the performance and scalability of producing unions of conjunctive queries versus datalog queries as rewritings. We present an empirical comparison between two representative approaches that consider very expressive ontology languages.

1 Introduction

The use of ontologies for query answering allows for the extraction of both explicit and *implicit* knowledge from the underlying data. Query answering over ontologies is a crucial problem in contexts such as ontology-based data access [12] and semantic information integration [10].

The main query language considered in the literature is that of *conjunctive queries* (which captures the core of SPARQL queries). In contrast, several ontology languages of various levels of expressivity have been considered. Query answering for very expressive languages such as OWL 2 DL is known to be intractable [7]. Fortunately, three *profiles* of OWL 2 with good computational properties have been identified: QL, RL, and EL.[1] In fact, query answering over QL ontologies is known to be only as hard as evaluating SQL queries over a relational database [3].[2]

Query answering in QL can be performed via *query rewriting* in two steps: first, the query and the *terminological* part of the ontology (i.e., the schema or TBox) are transformed into a so-called *rewriting*; and then the rewriting is evaluated over the *assertional* part of the ontology (i.e., the data or ABox) only.

[1] http://www.w3.org/TR/owl2-profiles/
[2] With respect to data complexity.

In this case, the rewriting is an expanded version of the original query in the form of a union of conjunctive queries (UCQ). Therefore, reasoners implementing query rewriting not only avoid keeping potentially very large ABoxes in memory, but may delegate evaluation of the rewriting to off-the-shelf, highly optimized RDBMSs.

Various UCQ-producing rewriting algorithms have been devised for (variants of) QL [3, 11, 15, 5, 8]; alas, the size of the rewritings has been shown to be exponential with respect to the size of the original query and the TBox [3]. In practice, this means that the computed rewriting might contain hundreds or thousands of queries, rendering it too big to evaluate efficiently (or at all) over existing technology. In order to address this problem, alternative approaches have been devised [16, 6] in which, instead of producing a potentially large UCQ, the original query and the TBox are rewritten into a more succinct *datalog query* (DQ). Datalog queries, however, are harder to evaluate than UCQs [1], which suggests there is a trade-off between the size of the rewriting and the complexity of its evaluation.

In this paper, we consider the advantages and disadvantages of producing DQs over UCQs in terms of scalability of query answering. We begin by presenting the query rewriting approach in more detail in Section 2. We then present a general overview of existing approaches (of the two kinds) in Section 3. The main contribution of the paper is an empirical evaluation in which we compare the (DQ-producing) approach of Eiter et al. [6] against Blackout—a highly optimized version of the (UCQ-producing) approach of Pérez-Urbina et al. [11]. We present Blackout in more detail in Section 4. The results of our evaluation are presented in Section 5. We present our conclusions in Section 6, and discuss our plans for future work in Section 7.

2 Query Answering via Rewriting

In this section, we introduce the notion of query rewriting informally by means of an example; we then discuss the advantages and disadvantages of the approach, and we finish with relevant formal definitions.

Query rewriting is a technique that can be used to solve the problem of query answering over ontologies—that is, given a *conjunctive* query and an ontology, composed of a TBox and an ABox, compute the set of *certain answers* of the query with respect to the ontology. The main idea behind query rewriting is to transform the given query and *TBox* into an expanded query that can be later evaluated over the ABox *only*. Intuitively, the expanded query contains all the relevant information captured in the TBox, making the latter unnecessary for query evaluation.

Example 1. Suppose we have an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ that talks about universities, students, professors, and so on. The TBox $\mathcal{T}$ contains the following axioms

(shown in Manchester syntax[3]):

`Class:` Teacher `SubClassOf:` teaches `some Thing` (1)

`Class:` Professor `SubClassOf:` Teacher (2)

`ObjectProperty:` hasTutor `Range:` Professor (3)

where axiom (1) states that teachers teach at least someone, axiom (2) states that professors are teachers, and axiom (3) states that all tutors are professors.

Suppose that we want to retrieve the list of individuals who teach according to $\mathcal{O}$ using the query Q (shown in datalog syntax [1]):

$$Q(x) \leftarrow \text{teaches}(x, y) \tag{4}$$

Before considering the data in $\mathcal{A}$, we can rewrite the query with respect to the TBox—that is, expand Q with the relevant knowledge in $\mathcal{T}$. According to the meaning of axioms (1)–(3), we conclude that all teachers, professors, and tutors teach; therefore, we expand Q with:

$$Q(x) \leftarrow \text{Teacher}(x) \tag{5}$$

$$Q(x) \leftarrow \text{Professor}(x) \tag{6}$$

$$Q(y) \leftarrow \text{hasTutor}(x, y) \tag{7}$$

We can now evaluate the resulting *union* of queries (4)–(7) over $\mathcal{A}$ without further consideration of $\mathcal{T}$.

The query rewriting approach has important advantages over the 'direct' query answering approach implemented in reasoners like Pellet,[4] HermiT,[5] or FaCT++.[6] Since the query and the TBox only are considered, reasoners implementing query rewriting need not maintain potentially large ABoxes in memory, a crucial feature for some applications in terms of scalability. Once the query has been rewritten, its evaluation can be delegated to existing highly optimized (deductive) database systems. Moreover, as the rewriting is independent from the ABox, one does not need to recompute it every time the data changes, but only when the TBox does. This is important in many application domains where data tends to change much more often than the schema.

The specification of OWL 2 includes the definition of various fragments or profiles that have been tailored with specific use cases in mind. In particular, the QL profile was designed so as to benefit from the advantages of query rewriting, both in terms of scalability and performance. It has been shown that queries posed over OWL 2 QL TBoxes can be rewritten into unions of conjunctive queries (UCQs) [3]. Producing UCQs is particularly desirable as their evaluation can be delegated to RDBMSs [12].

[3] `http://www.w3.org/TR/owl2-manchester-syntax/`

[4] `http://clarkparsia.com/pellet/`

[5] `http://hermit-reasoner.com/`

[6] `http://owl.man.ac.uk/factplusplus/`

Query rewriting is, alas, not a silver bullet. Depending of the nature of the expanded query, it might turn out to be too big and/or complex to evaluate efficiently (or at all). In particular, for instance, the size of a UCQ computed from a query and an OWL 2 QL TBox has been shown to be worst-case exponential (with respect to the size of the inputs) [3]. This means that we might compute a UCQ containing hundreds or thousands of queries, which would compromise the feasibility of its evaluation. Regarding complexity, once we consider more expressive fragments of OWL than QL, the resulting expanded query may not longer be a UCQ. Depending on how far we go with respect to ontology expressivity, we might need to produce *recursive* or even *disjunctive* datalog queries in order to ensure the soundness and completeness of the results. As one might expect, these types of query are harder to evaluate than UCQs. In such cases, one needs a more sophisticated machinery, such as that implemented in a deductive database system, for query evaluation.

An alternative approach to query rewriting is a technique based on forward chaining [4], known as *materialization*. This approach consists of expanding the ABox, instead of the query, with respect to the TBox, to effectively make all the implicit knowledge explicit. This approach might be preferable to query rewriting in cases where there are no changes to the ontology; queries may be executed as they are, without the need to rewrite them into potentially larger, more difficult to answer ones. Materialization, however, has significant drawbacks when changes to the data are frequent. This is due to the fact that materialized inferences need to be maintained in order for query answering to remain sound and complete. Thus, materialization may not be efficient in domains where data changes frequently. In contrast, query rewritings are independent of the ABox; therefore, one does not need to recompute them every time the data changes, but only when the TBox does. Materialization may also not be feasible as the expanded ABox might be prohibitively large. In contrast, query rewriting requires no modifications to the ABox.

Other alternatives include hybrid approaches where both the query *and* the ABox are expanded with respect to the TBox. The objective of these approaches is to avoid the potential exponential explosion in the size of the expanded query by using certain types of axiom to expand the ABox, while still retaining a manageable size. Unfortunately, similarly to materialization, these approaches might not be very efficient in scenarios where the data changes frequently.

We finish this section by giving a formal definition of the various notions described thus far. We use the well-known notions of constants, variables, function symbols, terms, and atoms of first-order logic [4].

Definition 1. *A* Horn clause *is an expression of the form* $H \leftarrow B_1 \wedge \cdots \wedge B_m$*, where* H *is a possibly empty atom and* $\{B_i\}$ *is a set of atoms. The atom* H *is called the* head *and the set* $\{B_i\}$ *is called the* body*. A Horn clause* C *is* safe *if all variables occurring in the head also occur in the body.*

A datalog *program* P *is a set of function-free, safe Horn clauses. The* extensional database (EDB) predicates *of* P *are those that do not occur in the head atom of any Horn clause in* P*; all other predicates are called* intensional

database (IDB) predicates. *A* datalog query *(DQ) Q is a tuple $\langle Q_P, P\rangle$, where Q_P is a* query predicate *and P is a datalog program. $Q = \langle Q_P, P\rangle$ is a* union of conjunctive queries *(UCQ) if Q_P is the only IDB predicate in P and the body of each clause in P does not contain Q_P, and Q is a* conjunctive query *(CQ) if it is a union of conjunctive queries and P contains exactly one Horn clause.*

An ontology *$\mathcal{O}$ is a tuple $\langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{T}$ is the terminological box or TBox, and $\mathcal{A}$ is the assertional box or ABox [2]. A tuple of constants $\vec{a}$ is an* answer *of a datalog query $Q = \langle Q_P, P\rangle$ on an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ if and only if $\mathcal{O} \cup P \models Q_P(\vec{a})$, where P is considered to be a set of universally quantified implications with the usual first-order semantics; the set of all answers of Q on $\mathcal{O}$ is denoted by $\mathsf{ans}(Q, \mathcal{O})$.*

Given a conjunctive query Q and a TBox $\mathcal{T}$, a datalog query $Q_\mathcal{T}$ is said to be a rewriting *of Q w.r.t. $\mathcal{T}$ if and only if $\mathsf{ans}(Q, \mathcal{T} \cup \mathcal{A}) = \mathsf{ans}(Q_\mathcal{T}, \mathcal{A})$ for every $\mathcal{A}$.*

3 State of the Art

The success of query rewriting depends on algorithms that produce manageable rewritings, both in terms of size and complexity. Since the seminal work of Calvanese et al. on DL-Lite—the Description Logic that provides the logical underpinning for OWL 2 QL—many rewriting algorithms aimed at efficient query answering via query rewriting have been proposed, most of which have been implemented in prototypes or commercial systems.

We limit ourselves to approaches implementing query rewriting as defined in Definition 1; that is, approaches where only the query gets expanded and the ABox is considered to be independent. Materialization-based and hybrid approaches—such as those by Kontchakov et al. [9], and Rodríguez-Muro and Calvanese [14]—are out of the scope of this paper.

Table 1. Overview of existing rewriting algorithms

	DL-Lite	Beyond DL-Lite
UCQ	PerfectRef Requiem Prexto Rapid Nyaya	Nyaya (Datalog$^{\pm}$)
DQ	Presto Clipper	Requiem ($\mathcal{ELHIO}^{\neg}$) Clipper (Horn-$\mathcal{SHIQ}$)

Notable approaches include that of Calvanese et al. (PerfectRef) [3], Pérez-Urbina et al. (Requiem) [11], Chortaras et al. (Rapid) [5], Rosati and Almatelli

(Presto) [16], Rosati (Prexto) [15], Gottlob et al. (Nyaya) [8], and Eiter et al. (Clipper) [6]. In Table 1 these approaches are classified by the type of rewritings they produce (either UCQ or DQ) and by the Description Logic (DL) they support. Unsurprisingly, most approaches have been proposed for DL-Lite; however, note that there are a few algorithms that support DL-Lite as well as more expressive logics (shown in parentheses). Among the approaches that go beyond DL-Lite, Requiem is the only one that produces UCQs for DL-Lite and DQs for more expressive logics. In fact, Requiem will only produce DQs when the rewriting *has* to be recursive in order to ensure correct results; therefore, in many cases Requiem will produce UCQs even for logics more expressive than DL-Lite.

Most of the papers cited previously include an empirical evaluation of the approach with respect to others. We have summarized these results in Table 2, which shows the comparison of the approaches with respect to the rewritings size (number of clauses), their structural complexity, the time it takes to compute them, and the time it takes to *evaluate* them over some ABox.

Table 2. Comparison of existing rewriting algorithms

Size	[Clipper $\approx$ Presto], Prexto $<$ [PerfectRef = Requiem = Rapid = Nyaya]
Complexity	[PerfectRef $\approx$ Requiem $\approx$ Prexto $\approx$ Rapid $\approx$ Nyaya] $<$ [Clipper $\approx$ Presto]
Time	Rapid, Nyaya, [Clipper $\approx$ Presto] $<$ Requiem $<$ PerfectRef
Eval time	[Requiem $\approx$ Clipper $\approx$ Presto] $<$ PerfectRef

Note 1. All these comparison were made over DL-Lite ontologies. The relationship between approaches separated with commas is not discussed in the literature.

The approaches that produce DQs (see Table 1) produce smaller rewritings than their UCQ-producing counterparts, with the exception of Prexto. This is due to the fact that UCQs are larger than semantically equivalent (non UCQ) DQs (conjunctive normal form versus disjunctive normal form). Prexto stands apart because, unlike other UCQ-producing approaches, it implements an optimization that considers the ABox to reduce the size of the rewritings. Regarding complexity, we see that UCQ-producing approaches do better than DQ-producing ones. With respect to time, as it is related to size, it is not surprising that producing DQs is faster than producing UCQs; it is important to mention, however, that among those algorithms that produce UCQs, some approaches are much more efficient than others (hours versus seconds). Finally, with respect to evaluation time, we see that Presto outperforms PerfectRef, and, interestingly, that Requiem, Clipper, and Presto perform similarly, in spite of the fact that Requiem's rewritings are larger.

The comparison between Requiem, Clipper, and Presto regarding evaluation times was carried out by evaluating the computed rewritings of each system using DLV.[7] Using such a system was necessary as both Presto and Clipper require a deductive database of this type for query evaluation even for DL-Lite ontologies; note, however, that Requiem produces UCQs in this scenario. Therefore, these results suggest that evaluating semantically equivalent UCQs and DQs in DLV takes approximately the same time, so there is no substantial gain in evaluation time by producing (smaller but more complex) DQs versus UCQs. It would be interesting to see, however, whether UCQs can be evaluated more efficiently in an RDBMS or an RDF database.

Another important aspect to consider is the time it takes to compute the rewritings. According to Table 2, both Clipper and Presto outperform Requiem with respect to this metric (in fact, Requiem is outperformed by every algorithm except PerfectRef). It would be interesting to see whether Requiem can be made faster by enhancing it with the various optimization techniques used in the other, more efficient approaches.

In order to address these two questions, we present an empirical evaluation of Clipper and Blackout—an optimized version of Requiem—in Section 5. We chose these two approaches as they are the ones that support the most expressive logic within their respective rewriting type (UCQ vs DQ) categories. The optimizations implemented in Blackout are described in Section 4.

4 Blackout Optimizations

In this section we describe Blackout, a highly optimized version of Requiem. Blackout is part of the state-of-the-art triplestore Stardog.[8] Besides careful software engineering for efficiency, Blackout improves Requiem with two core optimizations.

First, Blackout implements an eager query containment optimization, as opposed to Requiem's lazy approach that computes query containment as the last step. As observed in most of the papers referenced in Section 3, this is one of Requiem's major drawbacks as the final containment step could take a very long time. Eager containment prunes redundant queries earlier in the rewriting process; thus, it prevents additional rewritings from being generated from these redundant queries, which themselves would be redundant. Even though this does not ultimately change the number of rewritings, it does significantly minimize the time spent on query containment checks. Thus, Blackout does not waste time generating redundant queries that will eventually be pruned, and it reduces the total number of containment checks performed.

Second, Blackout implements an optimization technique known as *data oracle*. This optimization is related to the so-called extensional constraints technique presented in [13]. If a derived query contains an atom which is empty with respect to a given ABox, the query is discarded as it would obviously produce empty

[7] `http://www.dlvsystem.com/`
[8] `http://stardog.com/`

results when evaluated. For instance, consider the rewriting obtained in Example 1, if we knew via the *data oracle* that the class Professor had no instances in $\mathcal{A}$, then there would be no need to include query (6) in the final rewriting.

The effectiveness of this optimization lies in the fact that even when a TBox might contain a large number of classes and properties, the assertions in the ABox typically use a much smaller number of classes and properties. This is frequently the case for deep class hierarchies where asserted types use leaf classes of the hierarchy, instead of more general classes higher in the hierarchy. For this reason, querying for a generic class might produce many rewritings because of the class hierarchy, but we might not need to execute all of those rewritings depending on the specific ABox at hand.

In Section 2 we pointed out that one advantage of the query rewriting approach is that it is independent of the ABox, whereas clearly the data oracle optimization introduces a dependency. This dependency, however, is a very weak one and requires the data oracle to only check the existence of an atom, a class or a property, in the data. Stardog, like other RDF databases, maintains special index structures that make this very efficient. Therefore, the query rewriting component can still be loosely-coupled from the storage system and does not need to maintain special in-memory data structures for this optimization. Moreover, as will be discussed further, the data oracle implementation can be crucial to the success of query rewriting in practice.

5 Evaluation

In this section we present an empirical evaluation of Clipper and Blackout. Our evaluation is based on the LUBM benchmark—a well-known standard that provides customizable data generation capabilities.[9] We first examined the performance of the two approaches with respect to size/complexity of the rewritings, including the time it took to compute them; and then, we looked into how these rewritings perform when evaluated.

All experiments were performed on Ubuntu 3.0.0 with a 3.2Ghz AMD Phenom processor, 8GB of RAM running Java $1.6.0_3 3$ with 8GB allocated to the JVM for each run. We recorded the average of 10 runs after 5 warmups for each experiment.

5.1 Computing Rewritings

The first part of the evaluation consisted of rewriting the 14 LUBM queries with respect to three TBoxes: $\mathcal{T}_{\text{QL}}$, $\mathcal{T}_{\text{RL}}$, and $\mathcal{T}_{\text{EL}}$, which correspond to QL, RL, and EL versions of the LUBM TBox, respectively. Table 3 summarizes our results. On the left-hand-side, we show the time in milliseconds that each system took to produce the rewritings for the 14 queries, whereas on the right-hand-side we show the number of clauses that each system produced overall.

[9] `http://swat.cse.lehigh.edu/projects/lubm/`

Table 3. Computation of Rewritings

	Time (ms)			Size (clauses)		
	$\mathcal{T}_{\text{QL}}$	$\mathcal{T}_{\text{RL}}$	$\mathcal{T}_{\text{EL}}$	$\mathcal{T}_{\text{QL}}$	$\mathcal{T}_{\text{RL}}$	$\mathcal{T}_{\text{EL}}$
Clipper	8.4	4.3	14.0	145	138	866
Blackout	25.0	21.0	108.8	20	21	102

As can be seen, Blackout is generally slower than Clipper, but it produces smaller rewritings. We believe the rewritings with few clauses produced by Blackout are the result of the data oracle optimization. In order to verify the benefits of this optimization, we ran Blackout without it and obtained 66 clauses for $\mathcal{T}_{\text{QL}}$, 65 clauses for $\mathcal{T}_{\text{RL}}$, and 253 clauses for $\mathcal{T}_{\text{EL}}$. The most significant gain was in query 5 over $\mathcal{T}_{\text{EL}}$, in which Blackout produced a rewriting containing 51 clauses, whereas Blackout with no data oracle produced 117. These results suggest that the data oracle optimization, when applicable, may have a big impact.

As can be seen in Table 4, Clipper produced DQs exclusively for all the queries and profiles, whereas Blackout produced UCQs most of the time, even for RL and EL. These results suggest that it is not always necessary to produce DQs as rewritings even for RL or EL ontologies. The type of the rewriting impacts evaluation time as UCQs are structurally less complex than DQs, and might be easier to evaluate, depending on their size.

Table 4. Rewritings Type

	$\mathcal{T}_{\text{QL}}$	$\mathcal{T}_{\text{RL}}$	$\mathcal{T}_{\text{EL}}$
Clipper	DQ (100%)	DQ (100%)	DQ (100%)
Blackout	UCQ (100%)	UCQ (86%)	UCQ (79%)

5.2 Evaluating Rewritings

The second part of our evaluation consisted of evaluating the rewritings produced by the two approaches. We used DLV—a state-of-the-art deductive database system (datalog engine) maintained by DLVSYSTEM s.r.l.—to evaluate Clipper and Blackout rewritings, and we used Stardog to evaluate Blackout rewritings only.[10] We considered four ABoxes of increasing size: $\mathcal{A}_1$, $\mathcal{A}_{10}$, $\mathcal{A}_{100}$, and $\mathcal{A}_{1000}$, which contain approximately 138K, 1.38M, 13.8M, and 138M triples, respectively.[11]

[10] Note that Stardog cannot presently evaluate DQs.

[11] Since the current implementation of Clipper does not support data property assertions, we ignored this type of assertion in our tests.

Tables 5 and 6 summarize our results. Table 5 shows the time in milliseconds that it took DLV to evaluate the rewritings produced by Clipper and Blackout over the various ABoxes.

Table 5. DLV Evaluation Performance (ms)

	Clipper			Blackout		
	$\mathcal{T}_{\text{QL}}$	$\mathcal{T}_{\text{RL}}$	$\mathcal{T}_{\text{EL}}$	$\mathcal{T}_{\text{QL}}$	$\mathcal{T}_{\text{RL}}$	$\mathcal{T}_{\text{EL}}$
$\mathcal{A}_1$	23.3	19.9	24.0	15.2	15.5	17.4
$\mathcal{A}_{10}$	20.8	28.3	30.2	16.3	18.5	15.6
$\mathcal{A}_{100}$	-	-	-	-	-	-
$\mathcal{A}_{1000}$	-	-	-	-	-	-

As can be seen, DLV was able to execute Blackout rewritings faster than those of Clipper, which is not surprising as the former are smaller and less complex than the latter. Unfortunately, DLV was unable to execute the rewritings over $\mathcal{A}_{100}$ and $\mathcal{A}_{1000}$ due to lack of memory since it maintains all the clauses in memory. Clearly, in order to be able to evaluate rewritings and queries in general over large ABoxes, we would need a system that makes use of secondary storage.

Table 6. Stardog Evaluation Performance (ms)

	$\mathcal{T}_{\text{QL}}$	$\mathcal{T}_{\text{RL}}$	$\mathcal{T}_{\text{EL}}$
$\mathcal{A}_1$	1.3	1.0	0.9
$\mathcal{A}_{10}$	13.8	14.4	14.1
$\mathcal{A}_{100}$	133.5	137.2	133.2
$\mathcal{A}_{1000}$	334.0	1036.0	69558.0

Table 6 shows the time in milliseconds that it took Stardog to evaluate the rewritings produced by Blackout over the various ABoxes. Stardog allows the creation of in-memory and disk databases. The experiments over $\mathcal{A}_1$, $\mathcal{A}_{10}$, and $\mathcal{A}_{100}$ were carried out using in-memory databases, whereas those over $\mathcal{A}_{1000}$ were performed over a disk database. As can be seen, Stardog outperformed DLV both with respect to Clipper and Blackout rewritings regarding $\mathcal{A}_1$ and $\mathcal{A}_{10}$. Moreover, it was able to scale to $\mathcal{A}_{100}$ and $\mathcal{A}_{1000}$.

5.3 Computation and Evaluation

In this section we summarize the results from previous sections with respect to Blackout and Stardog. Table 7 shows the overall performance of Stardog in

milliseconds counting the time it took Blackout to compute the rewritings and the time it took Stardog to evaluate them. It also shows the percentage of time that was spent on evaluating the produced rewritings.

Table 7. Blackout/Stardog Overall Performance

	$\mathcal{T}_{\text{QL}}$		$\mathcal{T}_{\text{RL}}$		$\mathcal{T}_{\text{EL}}$	
	Eval	Total (ms)	Eval	Total (ms)	Eval	Total (ms)
$\mathcal{A}_1$	4.94%	26.3	4.54%	22.0	0.82%	109.7
$\mathcal{A}_{10}$	35.57%	38.8	40.64%	35.4	11.48%	122.9
$\mathcal{A}_{100}$	84.23%	158.5	86.71%	158.2	55.05%	242.0
$\mathcal{A}_{1000}$	93.04%	359.0	98.01%	1057.0	99.84%	69666.8

Our results show that the larger the ABox, the longer time is spent on evaluating the rewritings. Therefore, we believe it is important to produce the simplest and smallest rewritings possible, even if this means spending a bit more time on the rewriting phase.

6 Conclusions

In this section we present a summary of our results.

DQ-producing approaches, such as Clipper, do not necessarily produce smaller rewritings than UCQ-producing approaches as formerly thought. Taking into consideration the underlying data might result in optimizations, such as the data oracle optimization, that can significantly reduce the size of the produced UCQs. This type of optimization should apply to DQs as well; it would be interesting to see to what extent it does and the impact it has.

It is not necessary to produce DQs for RL and EL in many cases. This is important as users can benefit from several of the languages features that are not included in QL without needing a datalog engine for query evaluation. As shown in this paper, the size of UCQs can be reduced and, importantly, UCQs are amenable to straightforward parallel evaluation. Therefore, we believe that one should only have to deal with DQs, and datalog engines, when necessary.

Evaluating the rewritings dominates the overall query answering time at large scales. Therefore, even though it may not be beneficial at small scales, it is worth investing time on optimizing the computation of rewritings in order to produce smaller and simpler rewritings that can be evaluated more efficiently.

7 Future Work

We are currently working on adding datalog evaluation to Stardog so that it can correctly evaluate Blackout rewritings even when they are DQs. Our ongoing

implementation is based on the well-known algorithm Query-SubQuery [1]. Once it is ready, it will be interesting to compare its performance with that of DLV and other state-of-the-art datalog engines (e.g., IRIS[12]).

Additionally, we plan to work on the parallelization of UCQ evaluation within Stardog. Currently, Stardog executes the results of the query rewriting as a single query; it takes the UCQ produced by Blackout and creates a single query by unioning each of the queries. However, the queries composing the UCQ tend to be relatively small, simple, and easy to evaluate. Crucially, they are also independent: the results of one conjunct are not needed to produce the results of another. This lends itself very nicely to evaluation of each query in parallel, which can be done by taking advantage of existing architecture within Stardog.

We are also working on the implementation of SPARQL 1.1 with Stardog. There are new features in SPARQL 1.1 such as sub-queries and property paths that have an interesting overlap with features provided, or planned, within Stardog and Blackout. First, we will explore what the performance implications are for rewriting sub-queries in SPARQL 1.1, and if there are advantages the QSQ approach can provide during evaluation, or even if rewriting sub-queries is a feasible design. Additionally, with some OWL language features, such as transitivity, now available in SPARQL, we will determine whether Blackout can be used to handle queries utilizing these features.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. F. Baader and W. Nutt. *Basic Description Logics*, chapter 2, pages 47–100. Cambridge University Press, 2003.
3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning*, 9:385–429, 2007.
4. C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., Orlando, FL, USA, 1997.
5. A. Chortaras, D. Trivela, and G. B. Stamou. Optimized Query Rewriting for OWL 2 QL. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2011.
6. T. Eiter, M. Ortiz, M. Simkus, T.-K. Tran, and G. Xiao. Towards Practical Query Answering for Horn-SHIQ. In Y. Kazakov, D. Lembo, and F. Wolter, editors, *Description Logics*, volume 846 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.
7. B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive Query Answering for the Description Logic SHIQ. *CoRR*, abs/1111.0049, 2011.
8. G. Gottlob, G. Orsi, and A. Pieris. Ontological Queries: Rewriting and Optimization. In S. Abiteboul, K. Böhm, C. Koch, and K.-L. Tan, editors, *ICDE*, pages 2–13. IEEE Computer Society, 2011.

[12] `http://www.iris-reasoner.org/`

9. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in dl-lite. In F. Lin and U. Sattler, editors, *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR2010)*. AAAI Press, 2010.
10. M. Lenzerini. Data Integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.
11. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable Query Answering and Rewriting under Description Logic Constraints. *J. Applied Logic*, 8(2):186–209, 2010.
12. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking Data to Ontologies. *J. on Data Semantics*, X:133–173, 2008.
13. M. Rodriguez-Muro and D. Calvanese. Dependencies: Making Ontology Based Data Access Work. In P. Barceló and V. Tannen, editors, *AMW*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
14. M. Rodriguez-Muro and D. Calvanese. High performance query answering over dl-lite ontologies. In G. Brewka, T. Eiter, and S. A. McIlraith, editors, *KR*. AAAI Press, 2012.
15. R. Rosati. Prexto: Query Rewriting under Extensional Constraints in DL-Lite. In E. Simperl, P. Cimiano, A. Polleres, O. Corcho, and V. Presutti, editors, *ESWC*, volume 7295 of *Lecture Notes in Computer Science*, pages 360–374. Springer, 2012.
16. R. Rosati and A. Almatelli. Improving Query Answering over DL-Lite Ontologies. In F. Lin, U. Sattler, and M. Truszczynski, editors, *KR*. AAAI Press, 2010.