OptiRef: Query Optimization for Knowledge Bases
Wafaa El Husseini, Cheikh Brahim El Vaigh, François Goasdoué, Hélène Jaudoin

To cite this version:

HAL Id: hal-04023665
https://inria.hal.science/hal-04023665
Submitted on 10 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright
OptiRef: Query Optimization for Knowledge Bases

Wafaa El Husseini  
Univ. Rennes, CNRS, IRISA  
Lannion, France  
wafaa.el-husseini@irisa.fr

Cheikh-Brahim El Vaigh  
Univ. Bourgogne  
Dijon, France  
cheikh-brahim.el-vaigh@u-bourgogne.fr

François Goasdoué  
Univ. Rennes, CNRS, IRISA  
Lannion, France  
fg@irisa.fr

Hélène Jaudoin  
Univ. Rennes, CNRS, IRISA  
Lannion, France  
helene.jaudoin@irisa.fr

ABSTRACT
Ontology-mediated query answering (OMQA) consists in asking database queries on a knowledge base (KB); a KB is a set of facts, the KB’s database, described by domain knowledge, the KB’s ontology.

FOL-rewritability is the main OMQA technique: it reformulates a query w.r.t. the KB’s ontology so that the evaluation of the reformulated query on the KB’s database computes the correct answers. However, because this technique embeds the domain knowledge relevant to the query into the reformulated query, a reformulated query may be complex and its optimization is the crux of efficiency.

We showcase OptiRef that implements a novel, general optimization framework for efficient query answering on datalog±, description logic, existential rules, OWL and RDF/S KBs. OptiRef optimizes reformulated queries by rapidly computing, based on a KB’s database summary, simpler (contained) queries with the same answers. We demonstrate OptiRef’s effectiveness on well-established benchmarks: performance is significantly improved in general, up to several orders of magnitude in the best cases!

CCS CONCEPTS
• Information systems → Query optimization; Semantic web description languages; • Computing methodologies → Knowledge representation and reasoning.

KEYWORDS
Existential rules, query optimization, data summarization

ACM Reference Format:

1 MOTIVATIONS
Ontology-mediated query answering (OMQA), which consists in answering database queries on knowledge bases (KBs), has received considerable attention in the database, knowledge representation and semantic web fields [1]. OMQA is non-trivial because answers must be computed from the explicit data stored in the KB's database plus the implicit data logically entailed from the KB’s database using the KB’s ontology, i.e., the domain knowledge.

FOL-rewritability is the widely-investigated OMQA technique introduced in [6]. It reduces query answering on KBs to query answering on databases provided by DBMSs: a query \( q \) on a KB is reformulated w.r.t. the KB’s ontology \( O \) into a query reformulation \( q^\circ \), in which the knowledge from \( O \) relevant to \( q \) is embedded, then the answers to \( q \) are obtained by evaluating \( q^\circ \) on the KB’s database with a DBMS. FOL-rewritability has been studied for queries expressed as (union of) conjunctive queries (CQs); KBs expressed using datalog± and existential rules, description logics and OWL2, and RDF/S; query reformulations expressed as unions of CQs (UCQs) and non-recursive datalog programs (datalog*) that unfold to UCQs, unions of semi-CQs (USCQs), and joins of UCQs (JUCQs). Table 1 recalls these languages. Main related works are referenced in Table 2.

We argue that the crux of efficient FOL-rewritability is the optimization of query reformulations: a query reformulation may be complex and costly to evaluate [5, 22] because it logically models all the worst-case exponentially many – specializations (i.e., particular cases) of the query w.r.t. the ontology of the queried KB [6]. We therefore showcase OptiRef that implements a novel optimization framework for FOL-rewritability. It applies to all the works on FOL-rewritability in the literature (e.g., Table 2) by considering expressive KBs expressed using existential or datalog± rules. It optimizes query reformulations using a KB’s database summary into simpler (contained) queries with the same answers. We demonstrate on well-established benchmarks for DL-lite KBs, how OptiRef significantly improves query answering performance in general, up to several orders of magnitude. DL-lite is a practical fragment of datalog± and existential rules; it is the description logic that underpins the W3C’s OWL2 QL profile for OMQA on large KBs.

This paper is an in-depth study and generalization of preliminary ideas sketched in [3, 13].
2 ANSWERING QUERIES ON A KB

KBs. We consider KBs expressed with datalog± or existential rules. A KB is of the form $K = (O, D)$, where $O$ is the KB’s ontology and $D$ is the KB’s database. An ontology is a set of deductive rules of the form $\forall x_1(q_1(x_1) \rightarrow q_2(x_2))$, where $q_1$, $q_2$ are conjunctive queries (CQs, recall Table 1) with sets of answer variables $x_1, x_2$ respectively, such that $x_2 \subseteq x_1$ and $x_1$ is exactly the set of variables that occur in $q_1$. A database is a set of incomplete facts, i.e., within which existential variables model missing or unknown values.

Notation. We use small letters for constants, e.g., $f$, $h$, etc. and small italic letters for variables, e.g., $x$, $y$, etc. Also we omit existential variables in rules: existential variables are those that solely appear in the right-hand side of $\rightarrow$.

Example 2.1 (Running example). Let us consider the following DL-lite KB $K = (O, D)$:

$O = \{ r_1 : \text{ww}(x, y) \rightarrow \text{ww}(y, x), r_2 : \text{sup}(x, y) \rightarrow \text{ww}(x, y), r_3 : \text{PhD}(x) \rightarrow \text{sup}(y, x) \}$,
$D = \{(R(f), R(h), \text{sup}(f, w), \text{sup}(h, w), \text{PhD}(w), \text{ww}(f, h))\}$.

The ontology $O$ states: working with (ww) someone is a symmetrical relation (r1), supervising someone (sup) is a particular case of working with someone (r2), and PhD students (PhD) are necessarily supervised (r3). The database $D$ states: François (f) and Hélène (h), who are researchers (R), supervise Wafaa (w), who is a PhD student. Additionally, François works with Hélène.

Query answering. We consider queries from the languages shown in Table 1. The arity of a query $q$ is the cardinality of its set $\bar{x}$ of free or answer variables. A certain answer to a query $q(x)$ of arity $n$ on a KB $K$ is a tuple of $n$ constants $\bar{t}$ such that $K \models q(\bar{t})$, where $q(\bar{t})$ is the Boolean query obtained by instantiating $\bar{x}$ with $\bar{t}$ in $q$. From now, we denote $\text{ans}(q, K)$ the answer set of $q$ on $K$.

Example 2.2 (Cont.). Let us consider the CQ asking for the supervisors who work with the researcher $h$: $q(x) = \exists y R(h) \wedge \text{ww}(h, x) \wedge \text{sup}(y, x)$. Its answer set on $K$ is $\text{ans}(q, K) = \{w\}$: $w$ can be obtained from $R(h) \in D$, $\text{sup}(f, w) \in D$ and the entailed fact $\text{ww}(w, w)$ deduced from $\text{sup}(h, w) \in D$ and $r_2$.

Notation. We omit the existential quantifiers to simplify the notation of queries, because non-answer variables are necessarily existentially quantified in the query languages we consider (recall Table 1). For instance, the CQ $q(x) = \exists y R(h) \wedge \text{ww}(h, x) \wedge \text{sup}(y, x)$ of Example 2.2 is now written $q(x) = R(h) \wedge \text{ww}(h, x) \wedge \text{sup}(y, x)$.

FOL-rewritability. The main technique to perform query answering on KBs is FOL-rewritability. This technique is illustrated in Figure 1 using the black solid and dashed edges. The query $q$ is first reformulated based on the ontology $O$ into the query $q^O$ called a reformulation of $q$ w.r.t. $O$, so that the relational evaluation of $q^O$ on the database $D$, denoted by $\text{eval}(q^O, D)$, provides the correct answer set of $q$ on $K$, i.e., $\text{eval}(q^O, D) = \text{ans}(q, K)$.

The formal properties of FOL-rewritability, which motivated the optimizations we devised and we implemented in OptiRef, are [6]: (i) $q^O$ is equivalent to $q$ w.r.t. $O$ and encodes all the worst-case exponentially many non-redundant queries contained in $q$ w.r.t. $O$ and (ii) for any database $D'$, $\text{eval}(q^O, O') = \text{ans}(q, (O, D'))$ holds.

FOL-rewritability has been studied in the literature for triples of query, ontology and query reformulation languages (recall Table 1), which we term OMQA settings.

Example 2.3 (Cont.). The next UCQ $q^{\text{UCQ}}$, USCQ $q^{\text{USCQ}}$ and JUCQ $q^{\text{JUCQ}}$ are the reformulations of $q$ w.r.t. $O$ respectively computed by the Rapid [8], Compact [22] and GDL [5] tools.

$q^{\text{UCQ}}(x) = (R(h) \wedge \text{ww}(h, x) \wedge \text{sup}(y, x) \wedge \text{ww}(x, y) \wedge \text{PhD}(x))$ (1)
$\forall (R(h) \wedge \text{ww}(h, x) \wedge \text{PhD}(x))$ (2)
$\forall (R(h) \wedge \text{sup}(y, x))$ (3)
$\forall (R(h) \wedge \text{ww}(h, x) \wedge \text{PhD}(x))$ (4)
$\forall (R(h) \wedge \text{ww}(x, h) \wedge \text{PhD}(x))$ (5)
$\forall (R(h) \wedge \text{sup}(x, h) \wedge \text{sup}(y, x))$ (6)
$\forall (R(h) \wedge \text{sup}(x, h) \wedge \text{PhD}(x))$ (7)

$q^{\text{USCQ}}(x) = (R(h) \wedge \text{ww}(h, x) \wedge \text{sup}(y, x) \wedge \text{ww}(x, y) \wedge \text{PhD}(x))$ (8)

$q^{\text{JUCQ}}(x) = (R(h) \wedge \text{ww}(h, x) \wedge \text{sup}(y, x) \wedge \text{ww}(x, y) \wedge \text{PhD}(x))$ (9)

$q^{\text{UCQ}}$ is the union of all the non-redundant CQs contained in $q$ w.r.t. $O$. $q^{\text{USCQ}}$ and $q^{\text{JUCQ}}$ encode the same CQs up to the distributive property of $\wedge$ and $\lor$. The only answer to $q$ on $O$ (i.e., $w$), results from (3) in $q^{\text{UCQ}}$, shown in bold, as well as from the logical combination of the subqueries shown in bold in $q^{\text{USCQ}}$ and $q^{\text{JUCQ}}$ and from which (3) can be recovered by distributing the $\wedge$’s over the $\lor$’s.

Problem statement. We focus on the problem of optimizing FOL-rewritability in all the OMQA settings, at once.

3 REVISIING FOL-REWITRABILITY

FOL-rewritability under scrutiny. FOL-rewritability is by definition truly independent of the queried data: a query $q$ is reformulated into a query $q^O$ such that for any database $D$, $\text{ans}(q, (O, D))$ is obtained by the evaluation of $q^O$ on $D$. However, when answering a query $q$ on a particular KB $K = (O, D)$, the database $D$ at hand is just one of all the possible databases the reformulation $q^O$ accommodates to. We argue that this innate universality of $q^O$ limits the performance of its evaluation by a DBMS on a fixed database $D$: some queries contained in $q$ w.r.t. $O$ that $q^O$ encode may be irrelevant to $D$, i.e., have no answer on $D$, while evaluating them may take significant time. In Example 2.3, all the CQs in $q^{\text{UCQ}}$ but (3) are irrelevant to the database $D$, as well as in $q^{\text{USCQ}}$ and $q^{\text{JUCQ}}$, where they are encoded up to the distributive property of $\wedge$ and $\lor$.

Towards more efficient FOL-rewritability. We therefore revise the definition of FOL-rewritability so that a query is not just reformulated w.r.t. the KB’s ontology, but also w.r.t. the KB’s database, i.e., w.r.t. the whole KB. FOL-rewritability applies to an OMQA setting iff for any query $q$ and any KB $K = (O, D)$, there exists a query reformulation $q^K$ of $q$ w.r.t. $K$ such that (i) $q^K$ is contained in $q$
w.r.t. \( O \) and (ii) \( \text{eval}(q^K, D) = \text{ans}(q, K) \) holds. We note that when \( q^K \) is equivalent to \( q \) w.r.t. \( O \) (i.e., to \( q^O \)), a particular case of containment, the original and revised definitions of FOL-rewritability coincide. Our new definition essentially trades the universality of a query reformulation for its specificity to, we say its optimization for, a fixed database. It enables \( q^K \) to model not all the – worst-case exponentially many – queries contained in \( q \) w.r.t. \( O \) (condition (i)), but at least those that contribute to the answer set of \( q \) on \( K \) (condition (ii)). In Example 2.3, removing from \( q^\text{UCQ}, q^\text{USCQ}, q^\text{JUCQ} \) none, some or all of the irrelevant subqueries make them reformulations of \( q \) w.r.t. \( K \), the more subqueries irrelevant to \( D \) are removed, the more \( q^\text{UCQ}, q^\text{USCQ}, q^\text{JUCQ} \) are optimized for \( D \). To distinguish FOL-rewritability that only focuses on the ontology at hand, from its revision that focuses on the whole KB at hand, we term the former O-FOL-rewritability and the latter KB-FOL-rewritability.

4 OPTIMIZING QUERY REFORMULATIONS

OptiRef system. We devised the OptiRef system to turn all the query reformulation algorithms for O-FOL-rewritability from all the currently known OMQA settings into query reformulation algorithms for KB-FOL-rewritability. The technique it implements for this O- to KB-FOL-rewritability is illustrated in Figure 1 using the black and blue solid edges. Similarly to O-FOL-rewritability, the query \( q \) is first reformulated w.r.t. the ontology \( O \) into the reformulation \( q^O \) of \( q \) w.r.t. \( O \). This Reformulation step is achieved by using any off-the-shelf query reformulation algorithm for O-FOL-rewritability. Then, \( q^O \) is optimized for the database \( D \) to obtain the reformulation \( q^K \) of \( q \) w.r.t. \( K = (O, D) \). This Optimization step (discussed below) is achieved by rewriting \( q^K \), which is equivalent to \( q^O \), into \( q^K \), which is contained in \( q \), while retaining correctness of OMQA. \( q^K \) is simpler, hence faster to evaluate, than \( q^K \); \( q^K \) is obtained by removing subqueries irrelevant to \( D \) in \( q^O \). This optimization step uses a summary \( S \) of \( D \); \( S \) results from a Summarization step (discussed below). It is computed offline upon database loading and maintained upon database updates. Using \( S \) instead of \( D \) achieves a good trade-off between the number of identified subqueries irrelevant to \( D \) and the optimization time.

Summarization step (Fig. 1). We consider quotient databases as summaries, which we obtain by adapting the quotient operation from graph theory to our databases. This operation has been widely used for graph database summarization [7, 16]. It basically fuses equivalent nodes, database values in our case (i.e., constants and variables), according to an equivalence relation \( \equiv \). Quotient databases are homomorphic approximations of databases, hence enjoy an essential property for identifying CQs with no answer on a database. This property directly follows from the existence of a database-to-summary homomorphism: if \( q_a \) has no answer on \( S \), then \( q \) has no answer on \( D \), where \( q_a \) is obtained from \( q \) by replacing its constants by their images through the \( D \)-to-\( S \) homomorphism \( \sigma \).

Because ontology languages are centered on concepts modeled by unary relations, which are interrelated using relationships modeled by n-ary relations, our summarization is centered on the instances of concepts stored in the database \( D \). Our equivalence relation \( \equiv \) is such that (i) all the values that are instances of a same concept in \( D \) are represented by a single value in \( S \) and, to preserve joins within the data, (ii) for concepts whose instances share some value(s) in \( D \), all their values are represented by a same single value in \( S \).

Example 4.1 (Cont.). \( \equiv \) defines two equivalence classes of values in \( D \): \( \{f, h\} \) for the researchers and \( \{w\} \) for the PhD students. These two classes are respectively represented by \( p \) and \( r \) in the summary \( S = \{R(r), \text{sup}(r, p), \text{PhD}(p), \text{ww}(r, r), \text{ww}(w, p)\} \) of \( D \), with the \( D \)-to-\( S \) homomorphism \( \sigma \) such that \( \sigma(w) = p, \sigma(f) = \sigma(h) = r \).

Optimization step (Fig. 1). We devised a recursive optimization function \( \Omega \) that rewrites \( q^O \) into \( q^K \); \( \Omega \) identifies CQs with no answer on \( D \) using \( S \) and performs a bottom-up propagation of their removal within the query reformulation. \( \Omega \)'s base case is for CQs: \( \Omega(q, S) = \emptyset \) if \( q \) has no answer on \( S \) and is \( q \) itself otherwise, where \( \emptyset \) is the empty relation and \( \emptyset \) is the \( D \)-to-\( S \) homomorphism as seen above. \( \Omega \)'s first recursive case is for \( \wedge \): \( \Omega(\wedge_i^n q_i, S) = \emptyset \) if some \( \Omega(q_i, S) = \emptyset \) and is \( \wedge_i^n \Omega(q_i, S) \) otherwise. \( \Omega \)'s second recursive case is for \( \forall \): \( \forall(\forall_i^n q_i, S) = \emptyset \) if \( \Omega(q_i, S) = \emptyset \) for each \( i \) and is \( \forall_i^n \Omega(q_i, S) \) otherwise.

Crucially, \( q^K \) is a reformulation of \( q \) w.r.t. \( K \) because \( q^K = \Omega(q^O, S) \) and each subquery in \( q^O \) is recursively rewritten by \( \Omega \) into a contained one with same answer set on \( D \), hence \( q^K \) is contained in \( q \) w.r.t. \( O \) and eval(\( q^K, D \)) = \( \text{ans}(q, K) \) holds.

Example 4.2 (Cont.). The optimization of \( q^\text{UCQ}, q^\text{USCQ}, q^\text{JUCQ} \) for \( D \) using \( S \), which preserves the answer set, produces:

\[
\Omega(q^\text{UCQ}, S) = q^\text{USCQ}(x) = (R(h) \land \text{ww}(h, x) \land \text{sup}(y, x)) \lor (R(h) \land \text{ww}(h, x) \land \text{PhD}(x)) \lor (R(h) \land \text{sup}(h, x))
\]

\[
\Omega(q^\text{USCQ}, S) = q^\text{JUCQ}(x) = (R(h)) \land (ww(h, x) \lor \text{sup}(h, x) \lor \text{ww}(x, h)) \land (\text{sup}(y, x) \land \text{PhD}(x)) \lor (\text{ww}(x, h) \land \text{PhD}(x)) \lor (\text{sup}(h, x))
\]

where the subqueries in gray would have been additionally removed (with typically much longer optimization time) if the reformulations had been optimized by \( \Omega \) using \( D \) instead of \( S \).

5 OUTLINE OF THE DEMONSTRATION

We showcase OptiRef to highlight the time performance benefits that it brings to answering queries on KBs. We selected the practically relevant and reasonably expressive OMQA setting of CQs asked on DL-lite\(\Box \) KBs, as it enjoys: (i) FOL-rewritability based on UCQ, USCQ and JUCQ reformulations, (ii) well-established benchmarks, notably LUBM\(\Box \) [17] and NPD [15], and (iii) the W3C’s OWL2 QL profile to be used in real applications.

OptiRef is a JAVA tool that relies on the IBM DB2, MySQL and PostgreSQL DBMSs to store and query databases and their summaries, and on the Rapid [8], Compact [22] and GDL [5] JAVA tools to compute, respectively, UCQ, USCQ and JUCQ query reformulations of CQs w.r.t. DL-lite\(\Box \) ontologies. OptiRef also features a PHP/JQuery-based GUI to offer a nice visual and interactive attendee experience (see figures below). The demonstration first introduces O-FOL-rewritability, i.e., the state-of-the-art baseline, and motivates KB-FOL-rewritability based on Figure 1 and this paper’s simple examples. Then, the attendees can interact with OptiRef to experiment with O- and KB-FOL-rewritability on LUBM\(\Box \) and NPD KBs, and to assess and understand the time performance benefits that KB-FOL-rewritability brings.
The attendees can select and edit predefined queries or write their own queries (Fig. 2). The GUI indicates the query syntax to use and relations are chosen by browsing the LUBM \( \equiv \) NPD ontology with the Protégé ontology editor [18].

![Figure 2: GUI for asking queries to OptiRef](image)

To assess the performance improvements that OptiRef brings with its \( \Omega \) optimization function and summaries computed as database quotients with the \( \equiv \) equivalence relation, the attendees can compare the total query answering times to answer a query \( q \) on a KB \( K \) through the reformulation \( q^O \) of \( q \) w.r.t. \( K \)'s ontology \( O \), i.e., O-FOL-rewritability, and the optimization performed by \( \Omega \) of \( q^O \) into a reformulation \( q^K \) of \( q \) w.r.t. \( K \)'s database \( D \) or a summary \( S \) of it, i.e., KB-FOL-rewritability. The attendees can also observe how these times break down into reformulation, optimization and evaluation times (Fig. 3). They will learn that, for all the DBMSs, though to different extents, using \( D \) may improve or worsen performance, as optimization time may or may not amortize at evaluation time, while using \( S \) improves performance, significantly in general and up to several orders of magnitude.

![Figure 3: GUI for assessing OptiRef query answering performance: query answering times are shown for O-FOL-rewritability (left) and KB-FOL-rewritability when using the database (center) or its summary (right)](image)

To understand the time performance improvements, the attendees can visualize the logical plans and SQL of the reformulations in order to observe to which extent \( q^O \) has been optimized into \( q^K \) when \( \Omega \) uses \( D \) or \( S \). They can also compare the optimized reformulations w.r.t. the time to produce them (which depends on the DBMS at hand) and ratio of identified subqueries with no answer on \( D \) (Fig. 4): 100% when \( \Omega \) uses \( D \) and 80% on average when \( \Omega \) uses \( S \). The attendees can also browse OptiRef’s statistics to observe that the \( D \)-to-\( S \) size reduction is close to 90% for all the databases and that summarization time is linear in data size though visibly different from a DBMS to another (same code, data layout, indexes, etc.). They will learn that performance improvements follows from the use of summaries that, at the same time, are small enough for fast optimization time and allow a high ratio of identified subqueries with no answer for significant query reformulation optimization.

Acknowledgements. This work was supported by ANR CQFD (ANR-18-CE23-0003) and ARED Gedeon (LTC & Région Bretagne).

![Figure 4: GUI for examining OptiRef query reformulation optimization: the blue CQs are useful, i.e., they have answers, while the others are useless, i.e., they do not have answer; green ones are eliminated but red ones are not eliminated](image)

REFERENCES