Towards Top-k Query Answering in Description Logics: the case of DL-Lite

Umberto Straccia

ISTI - CNR, Pisa ITALY
straccia@isti.cnr.it

Abstract. We address the problem of evaluating ranked top-k queries in description logics. The problem occurs whenever we allow queries such as "find cheap hotels close to the conference location" in which fuzzy predicates like cheap and close occur. We show how to efficiently compute the top-k answers of conjunctive queries with fuzzy predicates over DL-LITE like knowledge bases.

1 Introduction

Description Logics (DLs) [2] provide popular features for the representation of structured knowledge. Nowadays, DLs have gained even more popularity due to their application in the context of the *Semantic Web*. DLs play a particular role as they are essentially the theoretical counterpart of state of the art languages to specify ontologies, such as *OWL DL* [12]. It becomes also apparent that in these contexts, data are typically very large and dominate the intentional level of the ontologies. Hence, while in the above mentioned contexts one could still accept reasoning that is exponential on the intentional part, it is mandatory that reasoning is polynomial in the data size, i.e. in *data complexity* [21]. Only recently efficient management of large amounts of data and its computational complexity analysis has become a primary concern of research in DLs and in ontology reasoning systems [1, 4, 5, 7, 11, 13].

In this paper we start addressing a novel issue for DLs with huge data repositories, namely the problem of *evaluating ranked top-k queries*. So far, an answer to a query is a set of tuples that satisfy a query. Each tuple does or does not satisfy the predicates in the query. However, very often the information need of a user involves so-called *fuzzy predicates* [22]. For instance, a user may need:"Find *cheap* hotels *near* to the conference location". Here, *cheap* and *near* are fuzzy predicates. Unlike the classical case, tuples satisfy now these predicates to a score (usually in $[0, 1]$). In the former case the score may depend, e.g., on the price, while in the latter case it may depend e.g. on the distance between the hotel location and the conference location. Therefore, a major problem we have to face with in such cases is that now an answer is a set of tuples *ranked* according to their *score*. This poses a new challenge in case we have to deal with a huge amount of instances. Indeed, virtually every tuple may satisfy a query with a non-zero score and, thus, has to be ranked. Computing all these scores, ranking them and then selecting the top-k ones is not feasible in practice, as we may deal with millions of tuples.

Our purpose is to address this problem for a DL-Lite like [4] description logic. DL-Lite has been specifically tailored to capture some basic ontology language features,

while keeping a low complexity of reasoning. Reasoning means not only computing the subsumption relationships between concepts, and checking satisfiability, but also answering complex queries (i.e. conjunctive queries) over a huge set of instances.

We extend DL-Lite by allowing fuzzy predicates to appear in the queries (we call the language DL-Lite) and propose methods to efficiently compute the top-k ranked answers. Similarly to DL-Lite, we may rely on existing techniques for top-k query answering developed in the context relational databases [6, 8, 16]. Furthermore, as for DL-Lite, query answering in DL-Lite is (sub) linear in data complexity, which makes the language appealing for real world scenarios.

We proceed as follows. In the next section we present DL-Lite. Then, we show how to efficiently compute the top-k answer set of conjunctive queries.

2 DL-Lite

As usual in DLs, *DL-Lite* allows for representing the domain of interest in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. Similarly to DL-Lite [1], in DL-Lite concepts and roles are defined as follows:

$$\begin{array}{l} B \rightarrow A \mid \exists R \mid B_1 \sqcap B_2 \mid B_1 \sqcup B_2 \\ C \rightarrow A \mid \bot \mid \exists R \mid C_1 \sqcap C_2 \\ R \rightarrow P \mid P^- \end{array}$$

where A denotes an atomic concept and P denotes an atomic role. A role R can be either an atomic role P or its *inverse* P^-. B denotes a *basic concept* that can be either an atomic concept, a concept of the form $\exists R$, i.e. the standard DL construct of unqualified existential quantification. C denotes a *general concept*. A DL-Lite *knowledge base* is pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ and $\mathcal{A}$ are finite sets of DL-Lite axioms and assertions. $\mathcal{T}$ is the TBox and is used to represent intentional knowledge, while $\mathcal{A}$ is the ABox and is used to represent extensional knowledge. An *axiom* is of the form $B \sqsubseteq C$ (inclusion axiom) and $\texttt{fun}(R)$ (functionality axiom). A functionality axiom expresses the functionality of a role. *Assertions* are of the form $B(a)$ (concept assertion) and $P(a, b)$ (role assertion). Assertions state the membership of an individual (resp. pair of individuals) to a basic concept (resp. role).

DL-Lite allows for querying the extensional knowledge of a KB by means of conjunctive queries of arbitrary complexity, where fuzzy predicates may appear. Furthermore, we allow disjunctive queries as well. A *conjunctive query* q over a knowledge base $\mathcal{K}$ is an expression of the form

$$q(\mathbf{x}, s) \leftarrow \exists \mathbf{y}.conj(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$$

where

1. $\mathbf{x}$ are the *distinguished variables*;
2. s is the *score variable*, taking values in $[0, 1]$;
3. $\mathbf{y}$ are existentially quantified variables called the *non-distinguished variables*;
4. $conj(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms of the form $A(z)$, or $P(z, z')$, where A and P are respectively an atomic concept and a role (but, not inverse role) in $\mathcal{K}$. z, z' are constants in $\mathcal{K}$ or variables in $\mathbf{x}$ or $\mathbf{y}$;

[1] See [5] for extensions of DL-Lite with their computational complexity analysis.

5. $\mathbf{z_i}$ are tuples of constants in $\mathcal{K}$ or variables in $\mathbf{x}$ or $\mathbf{y}$;
6. p_i is an n_i-ary *fuzzy predicate* assigning to each n_i-ary tuple $\mathbf{c}_i$ as *score* $p_i(\mathbf{c}_i) \in [0,1]$. We require that an n-ary fuzzy predicate p is *safe*, i.e. there is not an m-ary fuzzy predicate p' such that $m < n$ and $p = p'$. Informally, all parameters are needed in the definition of p;
7. f is a *scoring* function $f\colon [0,1]^n \to [0,1]$, which combines the scores of the n fuzzy predicates $p_i(\mathbf{c}_i)$ into an overall *query score* to be assigned to the score variable s. We assume that f is *monotone*, i.e., for each $\mathbf{v}, \mathbf{v}' \in [0,1]^n$ such that $\mathbf{v} \leq \mathbf{v}'$, $f(\mathbf{v}) \leq f(\mathbf{v}')$ holds, where $(v_1, \ldots, v_n) \leq (v'_1, \ldots, v'_n)$ iff $v_i \leq v'_i$ for all i. [2]

A *disjunctive query* $\mathbf{q}$ is a finite set of conjunctive queries in which all the rules have the same head.

Example 1. Suppose we have information about hotels and conferences. Assume we have a fuzzy predicate `close` measuring the closeness degree between hotels and conference locations, depending on the distance, and a fuzzy predicate `cheap`, which given the price determines how "cheap" a hotel is. We may ask to find cheap hotels close to a conference location, i.e. rank the hotels according to their degree of closeness and cheapness. We may represent the scenario in DL-Lite as follows.

$$\begin{aligned}
\texttt{Hotel} &\sqsubseteq \exists \texttt{HasHLoc.Location}\\
\texttt{Hotel} &\sqsubseteq \exists \texttt{HasHPrice.Price}\\
\texttt{Conference} &\sqsubseteq \exists \texttt{HasCLoc.Location}\\
\texttt{Hotel} \sqcap \texttt{Conference} &\sqsubseteq \bot
\end{aligned}$$

HasHLoc	
HotelID	HasLoc
h1	hl1
h2	hl2
⋮	⋮

HasCLoc	
ConfID	HasLoc
c1	cl1
c2	cl2
⋮	⋮

HasHPrice	
HotelID	Price
h1	150
h2	200
⋮	⋮

Then we may express our information need using the conjunctive query (`c1` is our conference location)

$$\begin{aligned}
q(h,s) \leftarrow &\texttt{HasHLoc}(h,hl) \wedge \texttt{HasHPrice}(h,p) \wedge\\
&\texttt{HasCLoc}(\texttt{c1},cl) \wedge s = \texttt{cheap}(p) \cdot \texttt{close}(hl,cl)\ .
\end{aligned}$$

where the fuzzy predicates `cheap` and `close` are defined as

$$\begin{aligned}
\texttt{close}(\texttt{hl},\texttt{cl}) &= \max(0, 1 - \tfrac{\texttt{distance(hl,cl)}}{2000})\\
\texttt{cheap}(\texttt{price}) &= \max(0, 1 - \tfrac{\texttt{price}}{300})
\end{aligned}$$

The `distance` function returns the distance between hotels and conferences, obtained from an external source, e.g. database relation or web page. Note that the scoring function is the product $f(\texttt{cheap}(p), \texttt{close}(hl,cl)) = \texttt{cheap}(p) \cdot \texttt{close}(hl,cl)$, which is monotone in its arguments `cheap` and `close`.

We want to retrieve the top-k answers according to the score s. Please note that it is not feasible to compute all scores first and then rank them (there may be a huge amount of hotels and conference locations).

[2] We assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant.

Note also that if we would like to find hotels, which are either cheap or close to the conference location, then we may use the disjunctive query:

$$q(h,s) \leftarrow \texttt{HasHPrice}(h,p) \wedge s = \texttt{cheap}(p)$$
$$q(h,s) \leftarrow \texttt{HasHLoc}(h,hl) \wedge \texttt{HasCLoc}(\texttt{c1},cl) \wedge s = \texttt{close}(hl,cl)$$

We point out that the above disjunctive query is different from the conjunctive query

$$q(h,s) \leftarrow \texttt{HasHLoc}(h,hl) \wedge \texttt{HasHPrice}(h,p) \wedge \\ \texttt{HasCLoc}(\texttt{c1},cl) \wedge s = \max(\texttt{cheap}(p), \texttt{close}(hl,cl))$$

as in the former we may find hotels, which are close to the conference location, though the price is unknown. □

Form a semantics point of view, it is similar to the usual semantics for DLs. The major difference is that we consider a *fixed infinite domain* Δ. [3] We assume to have one object for each constant, denoting exactly that object. In other words, we have standard names [15], and we will not distinguish between the alphabet of constants and Δ. So, an *interpretation* is a first-order structure $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ and consists of a fixed infinite domain Δ with an interpretation function $\cdot^{\mathcal{I}}$ such that:

$$A^{\mathcal{I}} \subseteq \Delta \qquad P^{\mathcal{I}} \subseteq \Delta \times \Delta \qquad \bot^{\mathcal{I}} = \emptyset$$
$$(C_1 \sqcap C_2)^{\mathcal{I}} = {C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}} \quad (P^-)^{\mathcal{I}} = \{\langle c,c'\rangle \mid \langle c',c\rangle \in P^{\mathcal{I}}\}$$
$$(C_1 \sqcup C_2)^{\mathcal{I}} = {C_1}^{\mathcal{I}} \cup {C_2}^{\mathcal{I}} \quad \exists R^{\mathcal{I}} = \{c \mid \exists c'.\langle c,c'\rangle \in R^{\mathcal{I}}\}$$

An interpretation $\mathcal{I}$ is a *model* of (i) an inclusion axiom $B \sqsubseteq C$ iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$; (ii) a functionality axiom $\texttt{fun}(R)$ iff $\langle c,c'\rangle \in R^{\mathcal{I}} \wedge \langle c,c''\rangle \in R^{\mathcal{I}} \Rightarrow c' = c''$; $(iiii)$ an assertion $B(a)$ (resp. $R(a,b)$) iff $a \in C^{\mathcal{I}}$ (resp. $(a,b) \in P^{\mathcal{I}}$); and (iv) a KB $\mathcal{K}$ iff $\mathcal{I}$ is a model of each axiom and assertion occurring in $\mathcal{K}$. A KB is *satisfiable* if it has a model. A KB $\mathcal{K}$ *entails* an assertion (resp. inclusion axiom) iff each model of the KB is also a model of the assertion (resp. inclusion axiom).

We recall that despite the simplicity of its language, the DL is able to capture the main notions (though not all, obviously) to represent structured knowledge. In particular, the axioms allow us to specify that concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$; *disjointness*, e.g., between concepts A_1 and A_2, using $A_1 \sqcap A_2 \sqsubseteq \bot$; *role-typing*, using $\exists P \sqsubseteq A_1$ and $\exists P^- \sqsubseteq A_2$; *participation constraints*, using $A \sqsubseteq \exists P$ and $A \sqsubseteq \exists P^-$; *non-participation constraints*, using $A \sqcap \exists P \sqsubseteq \bot$ and $A \sqcap \exists P^- \sqsubseteq \bot$. Additionally, observe that we allow cyclic axioms. Notice that DL-Lite is a strict subset of OWL Lite and, thus of OWL DL [12], which presents some constructs (e.g., some kinds of role restrictions) that are non expressible in DL-Lite(and that make reasoning in OWL Lite non-tractable in general).

Concerning queries, informally a conjunctive query $q(\mathbf{x},s) \leftarrow \exists \mathbf{y}.conj(\mathbf{x},\mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1),\ldots,p_n(\mathbf{z}_n))$ is interpreted in an interpretation $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\langle \mathbf{c}, v\rangle$, such that when we substitute the variables $\mathbf{x}$ and s with the constants $\mathbf{c} \in \Delta \times \ldots \times \Delta$ and the real value $v \in [0,1]$, the formula $\exists \mathbf{y}.conj(\mathbf{x},\mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1),\ldots,p_n(\mathbf{z}_n))$ evaluates to true in $\mathcal{I}$. But, due to the existential quantification $\exists \mathbf{y}$, for a fixed $\mathbf{c}$, there may be many substitutions $\mathbf{c}'$ for $\mathbf{y}$ and, thus, we may have many possible scores for the tuple $\mathbf{c}$. Among all these scores for $\mathbf{c}$, we select the highest one. Furthermore, if the

[3] The domain is infinite as pointed out in [4].

query is a disjunctive query $\mathbf{q}$, for each tuple $\mathbf{c}$ there may be a score v_i computed by each conjunctive query $q_i \in \mathbf{q}$. In that case, the overall score for $\mathbf{c}$ is the maximum among the scores v_i. Specifically, we assume that the score combination function f and the fuzzy predicates p_i have a given *fixed interpretation*. Now, let $\theta = \{\mathbf{x}/\mathbf{c}, \mathbf{y}/\mathbf{c}', s/v\}$ be a substitution of the variables $\mathbf{x}, \mathbf{y}$ and s with the tuples $\mathbf{c}, \mathbf{c}'$ and score value $v \in [0, 1]$. Let $\psi(\mathbf{x}, \mathbf{y}, s)$ be $conj(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$. With $\psi(\mathbf{x}, \mathbf{y}, s)\theta$ we denote the ground formula obtained by applying to $\psi(\mathbf{x}, \mathbf{y}, s)$ the substitution θ. We say that an interpretation $\mathcal{I}$ is a *model* of $\psi(\mathbf{x}, \mathbf{y}, s)\theta$ iff $\psi(\mathbf{x}, \mathbf{y}, s)\theta$ evaluates to true in $\mathcal{I}$. We will write $\mathcal{I} \models \psi(\mathbf{x}, \mathbf{y}, s)\theta$ in this case. Then, the interpretation $\mathbf{q}^{\mathcal{I}}$ of a disjunctive query $\mathbf{q} = \{q_1, \ldots, q_m\}$ in $\mathcal{I}$ is

$$\mathbf{q}^{\mathcal{I}} = \{\langle \mathbf{c}, v\rangle \in \Delta \times \ldots \times \Delta \times [0, 1] \mid v = \max(v_1, \ldots, v_m), \\ v_i = \sup_{\mathbf{c}' \in \Delta \times \ldots \times \Delta}\{v' \mid \mathcal{I} \models \psi_i(\mathbf{x}, \mathbf{y}, s)\theta'\}\},$$

where θ' is as θ, except that $\mathbf{y}$ is substituted with $\mathbf{c}'$ and s is substituted with v', each conjunctive query $q_i \in \mathbf{q}$ is of the form $q(\mathbf{x}, s) \leftarrow \exists \mathbf{y}.\psi_i(\mathbf{x}, \mathbf{y}, s)$, $\sup \emptyset$ is undefined, and $\max(v_1, \ldots, v_n)$ is undefined iff all its arguments are undefined. Therefore, some tuples $\mathbf{c}$ may not have a score in $\mathcal{I}$ and, thus, $\langle \mathbf{c}, v\rangle \notin \mathbf{q}^{\mathcal{I}}$ for no $v \in [0, 1]$. Alternatively we may define $\sup \emptyset = 0$ and, thus, all tuples $\mathbf{c}$ have a score in $\mathcal{I}$, i.e. $\langle \mathbf{c}, v\rangle \in \mathbf{q}^{\mathcal{I}}$ for some $v \in [0, 1]$. We use the former formulation to distinguish the case where a tuple $\mathbf{c}$ is retrieved, though the score is 0, from the tuples which do not satisfy the query and, thus, are not retrieved. Finally, for all $\mathbf{c} \in \Delta \times \ldots \times \Delta$ and for all $v \in [0, 1]$, we say that $\mathcal{I}$ is a *model* of $q(\mathbf{c}, v)$ iff $\langle \mathbf{c}, v\rangle \in \mathbf{q}^{\mathcal{I}}$. Also, we say that a satisfiable KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ *entails* $q(\mathbf{c}, v)$, written $\mathcal{K} \models q(\mathbf{c}, v)$ iff any model $\mathcal{I}$ of $\mathcal{K}$ is also model of $q(\mathbf{c}, v)$ (note that $\mathcal{K}$ is required to be satisfiable).

The basic reasoning services that mainly concerns us is the knowledge base satisfiability problem and the top-k retrieval problem, where this latter is defined as:

Top-k retrieval: Given a satisfiable KB $\mathcal{K}$, retrieve the top-k ranked tuples $\langle \mathbf{c}, v\rangle$ that instantiate the disjunctive query $\mathbf{q}$ and rank them in decreasing order w.r.t. the score, i.e. find the top-k ranked tuples of the answer set of $\mathbf{q}$, denoted $ans_k(\mathcal{K}, \mathbf{q}) = \mathtt{Top}_k\{\langle \mathbf{c}, v\rangle \mid \mathcal{K} \models q(\mathbf{c}, v)\}$.

Some comments are in order on the form of the queries. Overall, our language extension to classical DLs, such as DL-Lite, concerns only the query language part and not the data representation language (which remains a classical DLs). This is exactly as it happens in top-k retrieval in the context of relational databases [6, 8, 16]: the data is represented as usual in relational tables and the SQL query language is extended to allow to express a scoring function as well, which may use the values occurring in the retrieved records, to compute the score of the record. By referring to Example 1, one may naturally ask why we do not allow to represent a fuzzy concept such as "cheap hotel" in the language and associate to each instance of it a score, as it happens usually in fuzzy DLs [17, 19]. Besides a semantical shift from a classical semantics to a fuzzy one (and, thus, likely changing the kind of inferences allowed), we assume here that queries are not defined once for ever, but may be issued by users to the systems. This means that it is not feasible to compute all scores in advance, as the queries are not known a priory. Furthermore, even the data may be available on query time only, e.g. if it has to be gathered from the Web ("find a flat with a big living room"). It is thus not surprising that most work on top-k retrieval in relation databases focusses on minimizing the number of score function evaluations. What we will show here is that we can enhance query answering for classical DL-Lite with almost no additional effort.

In the following, for the sake of illustrative purposes, we consider the following abstract example.

Example 2. Suppose the set of inclusion axioms is $\mathcal{T} = \{\exists P_2^- \sqsubseteq A, A \sqsubseteq \exists P_1, B \sqsubseteq \exists P_2\}$. We also assume that the set of assertions $\mathcal{A}$ is stored in the three sets below (P_2 is a role, while B and C are basic concepts):

$$\begin{aligned} P_2 &= \{\langle 0, \mathtt{s}\rangle, \langle 3, \mathtt{t}\rangle, \langle 4, \mathtt{q}\rangle, \langle 6, \mathtt{q}\rangle\} \\ B &= \{\langle 1\rangle, \langle 2\rangle, \langle 5\rangle, \langle 7\rangle\} \\ C &= \{\langle 5\rangle, \langle 3\rangle, \langle 2\rangle, \langle 4\rangle\} \end{aligned}$$

Assume our disjunctive query is $\mathbf{q} = \{q', q''\}$ where q' is $q(x, s) \leftarrow \exists y \exists z. P_2(x, y) \wedge P_1(y, z) \wedge s = f(\mathtt{p}(x))$, q'' is $q(x, s) \leftarrow C(x) \wedge s = f(\mathtt{r}(x))$, the scoring function f is the identity $f(z) = z$ (f is monotone, of course), the fuzzy predicate $\mathtt{p}$ is $\mathtt{p}(x) = \max(0, 1 - x/10)$, and the fuzzy predicate $\mathtt{r}$ is $\mathtt{r}(x) = \max(0, 1 - (x/5)^2)$. Therefore, we can rewrite the query $\mathbf{q}$ as

$$\begin{aligned} q(x, s) \leftarrow & \exists y \exists z. P_2(x, y) \wedge P_1(y, z) \wedge s = \max(0, 1 - x/10) \\ q(x, s) \leftarrow & C(x) \wedge s = \max(0, 1 - (x/5)^2) \, . \end{aligned}$$

Now, it can be verified that $\mathcal{K} \models q(3, 0.7)$, $\mathcal{K} \models q(2, 0.84)$ and for any $v \in [0, 1]$, $\mathcal{K} \not\models q(9, v)$. In the former case, any model $\mathcal{I}$ of $\mathcal{K}$ satisfies $P_2(3, \mathtt{t})$. But, $\mathcal{I}$ satisfies $\mathcal{T}$, so $\mathcal{I}$ satisfies $\exists P_2^- \sqsubseteq \exists P_1$. As $\mathcal{I}$ satisfies $P_2(3, \mathtt{t})$, $\mathcal{I}$ satisfies $(\exists P_2^-)(\mathtt{t})$ and, thus, $(\exists P_1)(\mathtt{t})$. As $0.7 = \max(0, 1 - 3/10)$, it follows that $\langle 3, 0.7\rangle$ evaluates the body of q' true in $\mathcal{I}$. On the other hand, $\langle 3, 0.64\rangle$ evaluates the body of q'' true in $\mathcal{I}$. Hence, the maximal score for 3 is 0.7, i.e., $\mathcal{I}$ is a model of $q(3, 0.7)$. The other cases can be shown similarly. In summary, it can be shown that the top-4 answer set of $\mathbf{q}$ is $ans_4(\mathcal{K}, \mathbf{q}) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle, \langle 3, 0.7\rangle]$. □

3 Top-k query Answering

We discuss now how to determine the top-k answers of a disjunctive query over a DL-Lite knowledge base $\mathcal{K}$. To this end:

1. We have to check if $\mathcal{K}$ is satisfiable, as querying a non-satisfiable KB is undefined in our case.
2. By considering $\mathcal{T}$ only, the user query $\mathbf{q}$ is *reformulated* into a set of conjunctive queries $r(\mathbf{q}, \mathcal{T})$. Informally, the basic idea is that the reformulation procedure closely resembles a top-down resolution procedure for logic programming, where each inclusion axiom $B_1 \sqsubseteq B_2$ is seen as a logic programming rule of the form $B_2(x) \leftarrow B_1(x)$. For instance, given the query $q(x, s) \leftarrow A(x) \wedge s = f(\ldots)$ and suppose that $\mathcal{T}$ contains the inclusion axioms $B_1 \sqsubseteq A$ and $B_2 \sqsubseteq A$, then we can reformulate the query into two queries $q(x, s) \leftarrow B_1(x) \wedge s = f(\ldots)$ and $q(x, s) \leftarrow B_2(x) \wedge s = f(\ldots)$, exactly as it happens for top-down resolution methods in logic programming.
3. The reformulated queries in $r(\mathbf{q}, \mathcal{T})$ are *evaluated* over $\mathcal{A}$ only (which is stored in a database), producing the requested top-k answer set $ans_k(\mathcal{K}, \mathbf{q})$. For instance, for the previous query, the answers will be the top-k answers of the union of the answers produced by all three queries.

A feature of DL-Lite is that satisfiability checking and query reformulation is as for DL-Lite [4] [4], while the top-k evaluation step is novel. For the sake of completeness of the paper, we start with step 1 and step 2 above. So, we start by preparing our knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ for effective management. That means, we first normalize it into a suitable form and then store the data in $\mathcal{A}$ into a relational database.

KB normalization. The *normalization* of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is obtained by transforming $\mathcal{K}$ as follows. $\mathcal{A}$ is expanded by adding to $\mathcal{A}$ the assertions $(\exists P)(a)$ and $(\exists P^-)(b)$ for each $P(a, b) \in \mathcal{A}$. Concerning $\mathcal{T}$, we split concept conjunctions using the rule: if $\mathcal{T}$ contains $B \sqsubseteq C_1 \sqcap C_2$, then replace it with the two axioms $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$. Similarly, we split concept disjunctions: if $\mathcal{T}$ contains $B_1 \sqcup B_2 \sqsubseteq C$, then replace it with the two axioms $B_1 \sqsubseteq C$ and $B_2 \sqsubseteq C$. Furthermore, let $\sqsubseteq^*$ be the reflexive and transitive closure of the $\sqsubseteq$ relation over the roles inclusion axioms in $\mathcal{T}$.

Now $\mathcal{T}$ contains role inclusion axioms, functionality axioms and concept inclusion axioms of the form $B \sqsubseteq C$, according to the syntax rules

$$B \rightarrow A \mid \exists R \mid B_1 \sqcap B_2$$
$$C \rightarrow A \mid \bot \mid \exists R\,.$$

Then $\mathcal{T}$ is expanded by closing it with respect to the following inference rule: if $B_1 \sqcap C \sqsubseteq \bot$ occurs in $\mathcal{T}$ and $B_2 \sqsubseteq C$ occurs in $\mathcal{T}$, then add $B_1 \sqcap B_2 \sqsubseteq \bot$ to $\mathcal{T}$ (the rule applies also if B_1 is omitted).

It is easy to show that the normalization process transforms $\mathcal{K}$ into a model preserving form. In the following, without loss of generality we assume that every concept name or role name occurring in $\mathcal{A}$ also occurs in $\mathcal{T}$. Now we store $\mathcal{A}$ in a relational database. That is, (i) for each basic concept B occurring in $\mathcal{A}$, we define a relational table tab_B of arity 1, such that $\langle a \rangle \in tab_B$ iff $B(a) \in \mathcal{A}$; and (ii) for each role P occurring in $\mathcal{A}$, we define a relational table tab_P of arity 2, such that $\langle a, b \rangle \in tab_P$ iff $P(a, b) \in \mathcal{A}$. We denote with $\mathtt{DB}(\mathcal{A})$ the relational database thus constructed.

KB satisfiability. To check the satisfiability of a normalized KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we verify the following conditions, which can easily be derived from [4]: (i) there exists $B \sqsubseteq \bot \in \mathcal{T}$ and a constant a such $B(a) \in \mathcal{A}$; (ii) there exists $B_1 \sqcap B_2 \sqsubseteq \bot \in \mathcal{T}$ and a constant a such $\{B_1(a), B_2(a)\} \subseteq \mathcal{A}$; (iii) there exists an axiom $\mathtt{fun}(P)$ (respectively, $\mathtt{fun}(P^-)$) in $\mathcal{T}$ and three constants a, b, c such that both $P(a, b)$ and $P(a, c)$ (resp., $P(b, a)$ and $P(c, a)$) belong to $\mathcal{A}$; If one of the conditions above holds, then $\mathcal{K}$ is not satisfiable. Otherwise, $\mathcal{K}$ is satisfiable. Note that the algorithm can verify such conditions by posing to $\mathtt{DB}(\mathcal{A})$ simple SQL queries.

Query reformulation. The query reformulation step is adapted from [4] to our case and is as follows. We say that a variable in a conjunctive query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body (inclusive the scoring function), or a constant, while we say that a variable is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, we use the symbol "_" to represent non-distinguished non-shared variables). Note that an atom of the form $(\exists P)(x)$ (resp. $(\exists P^-)(x)$) has the same meaning as $P(x, \_)$ (resp. $P(\_, x)$). For ease of exposition, in the following we will use the latter form only. An axiom τ is *applicable* to an atom $B(x)$, if τ has B in its right-hand side, and τ is applicable to an atom $P(x_1, x_2)$, if either (i) $x_2 = \_$ and the right-hand side of τ is $\exists P$,

[4] Strictly speaking, DL-Lite does not support disjunctive queries, though it can easily be extended to that case.

or (ii) $x_1 = \_$ and the right-hand side of τ is $\exists P^-$. We indicate with $gr(g;\tau)$ the atom obtained from the atom g by applying the inclusion axiom τ. Specifically, if $g = B_1(x)$ (resp., $g = P_1(x,\_)$ or $g = P_1(\_,x)$) and $\tau = B_2 \sqsubseteq B_1$ (resp., $\tau = B_2 \sqsubseteq \exists P_1$ or $\tau = B_2 \sqsubseteq \exists P_1^-$), we have:

- $gr(g,\tau) = A(x)$, if $B_2 = A$, where A is an atomic concept;
- $gr(g,\tau) = P_2(x,\_)$, if $B_2 = \exists P_2$;
- $gr(g,\tau) = P_2(\_,x)$, if $B_2 = \exists P_2^-$;
- $gr(g,\tau) = B_3(x) \wedge B_4(x)$, if $B_2 = B_3 \sqcap B_4$.

We are now ready to present the query reformulation algorithm. Given a disjunctive query $\mathbf{q}$ and a set of axioms $\mathcal{T}$, the algorithm reformulates $\mathbf{q}$ in terms of a set of conjunctive queries $r(\mathbf{q},\mathcal{T})$, which then can be evaluated over $\mathtt{DB}(\mathcal{A})$. In the algo-

Algorithm 1 $\mathtt{QueryRef}(\mathbf{q},\mathcal{T})$

Input: Disjunctive query $\mathbf{q}$, DL-Lite axioms $\mathcal{T}$.
Output: Set of reformulated conjunctive queries $r(\mathbf{q},\mathcal{T})$.

```
1: r(q, T) := q
2: repeat
3:     S = r(q, T)
4:     for all q ∈ S do
5:         for all g ∈ q do
6:             if τ ∈ T is applicable to g then
7:                 r(q, T) := r(q, T) ∪ {q[g/gr(g, τ)]}
8:         for all g1, g2 ∈ q do
9:             if g1 and g2 unify then
10:                r(q, T) := r(q, T) ∪ {κ(reduce(q, g1, g2))}
11: until S = r(q, T)
12: r(q, T) := removeSubs(r(q, T))
13: return r(q, T)
```

rithm, $q[g/g']$ denotes the query obtained from q by replacing the atom g with a new atom g'. At step 8, for each pair of atoms g_1, g_2 that unify, the algorithm computes the query $q' = reduce(q, g_1, g_2)$, by applying to q the most general unifier between g_1 and g_2 [5]. Due to the unification, variables that were bound in q may become unbound in q'. Hence, inclusion axioms that were not applicable to atoms of q, may become applicable to atoms of q' (in the next executions of step (5)). Function κ applied to q' replaces with $\_$ each unbound variable in q'. Finally, in step 12 we remove from the set of queries $r(\mathbf{q},\mathcal{T})$, those which are already subsumed in $r(\mathbf{q},\mathcal{T})$. The notion of query subsumption is similar as for the classical database theory [20]. Given two queries q_i $(i = 1, 2)$ with same head $q(\mathbf{x}, s)$ and $q_1 \neq q_2$, we say that q_1 *is subsumed by* q_2, denoted $q_1 \sqsubseteq q_2$, iff for any interpretation $\mathcal{I}$, ${q_1}^{\mathcal{I}} \preceq {q_2}^{\mathcal{I}}$, where this latter is defined as: ${q_1}^{\mathcal{I}} \preceq {q_2}^{\mathcal{I}}$ iff for all $\langle \mathbf{c}, v_1 \rangle \in {q_1}^{\mathcal{I}}$ there is $\langle \mathbf{c}, v_2 \rangle \in {q_2}^{\mathcal{I}}$ such that $v_1 \leq v_2$. Essentially, if $q_1 \sqsubseteq q_2$ and both q_1 and q_2 belong to $r(\mathbf{q},\mathcal{T})$ then we can remove q_1 from $r(\mathbf{q},\mathcal{T})$ as q_1 produces a lower ranked result than q_2 with respect to the same tuple $\mathbf{c}$. In order to decide query subsumption, we can take advantage of the results in [14], related to the query containment part. A condition for query subsumption is the following. Assume that q_1 and q_2 do not share any variable. This can be accomplished by renaming all variables in e.g. q_1. Then it can be shown that

[5] We say that two atoms $g_1 = r(x_1,\ldots,x_n)$ and $g_2 = r(y_1,\ldots,y_n)$ *unify*, if for all i, either $x_i = y_i$ or $x_i = \_$ or $y_i = \_$. If g_1 and g_2 unify, then the unification of g_1 and g_2 is the atom $r(z_1,\ldots,z_n)$, where $z_i = x_i$ if $x_i = y_i$ or $y_i = \_$, otherwise $z_i = y_i$ [3].

Proposition 1 *If q_1 and q_2 share the same score combination function, then $q_1 \sqsubseteq q_2$ iff there is a variable substitution θ such that for each predicate $P(\mathbf{z}_2)$ occurring in the rule body of q_2 there is a predicate $P(\mathbf{z}_1)$ occurring in the rule body of q_1 such that $P(\mathbf{z}_2) = P(\mathbf{z}_1)\theta$.*

More complicated are cases in which q_1 and q_2 do not share the same score combination function. For instance, given

$$\begin{array}{ll} q_1 := q(x_1, s_1) \leftarrow & P(x_1, y_1) \wedge s_1 = y_1 \\ q_2 := q(x_2, s_2) \leftarrow & P(x_2, y_2) \wedge s_2 = \min(1, (x_2 + y_2)/2) \\ q_3 := q(x, s) \quad \leftarrow & P(x, y) \wedge s = \min(1, x + y) \end{array}$$

It can be shown that $q_2 \sqsubseteq q_3$ is the only subsumption relation among the queries above. Note that there is a variable substitution $\theta_{23} = \{x_2/x, y_2/y, s_2/s\}$ such that $P(x, y) = P(x_2, y_2)\theta_{23}$ and $\min(1, (x_2 + y_2)/2)\theta_{23} \leq \min(1, x + y)$, for all x, y. On the other hand, $q_1 \not\sqsubseteq q_2$. Note that we can find $\theta_{12} = \{x_1/x_2, y_1/y_2, s_1/s_2\}$ such that $P(x_2, y_2) = P(x_1, y_1)\theta_{12}$. However, $y_1\theta_{12} = y_2 \not\leq \min(1, (x_2 + y_2)/2)$, for all x_2, y_2. Similarly, $q_2 \not\sqsubseteq q_1$ and we can find $\theta_{21} = \{x_2/x_1, y_2/y_1, s_2/s_1\}$ such that $P(x_1, y_1) = P(x_2, y_2)\theta_{21}$ with $\min(1, (x_2 + y_2)/2)\theta_{21} = \min(1, (x_1 + y_1)/2) \not\leq y_1$, for all x_1, y_1. Hence, we can extend the query subsumption condition in Proposition 1 in the following way. Let q_1 and q_2 be two queries with same head and let σ_1 and σ_2 be the scoring component of q_1 and q_2, respectively. Then it can be shown that

Proposition 2 *$q_1 \sqsubseteq q_2$ iff there is a variable substitution θ such that for each predicate $P(\mathbf{z}_2)$ occurring in the rule body of q_2 there is a predicate $P(\mathbf{z}_1)$ occurring in the rule body of q_1 such that $P(\mathbf{z}_2) = P(\mathbf{z}_1)\theta$, and $\sigma_1\theta \leq \sigma_2$ for all variables occurring in $\sigma_1\theta$ and σ_2.*

Of course, the complexity of checking a condition such as $\sigma_1\theta \leq \sigma_2$ depends on the scoring functions and fuzzy predicates involved, and may be computationally expensive. We will not analyze this issue further in this paper and, thus, we assume that procedure $removeSubs$ removes subsumed queries according to Proposition 1, which is easy to check and not time consuming (we also could be more specific in the query subsumption definition, by restricting the interpretations to the models of the knowledge base, but this may lead to a query containment checking algorithm requiring a non-negligible amount of time). This concludes the query reformulation step.

Example 3. Consider Example 2. At step 1 $r(\mathbf{q}, \mathcal{T})$ is initialized with $\{q', q''\}$. It is easily verified that both conditions in step 6 and step 9 fail for q''. So we proceed with q'. Let σ be $s = \max(0, 1 - x/10)$. Then at the first execution of step 7, the algorithm inserts query q_1, $q(x, s) \leftarrow P_2(x, y) \wedge A(y) \wedge \sigma$ into $r(\mathbf{q}, \mathcal{T})$ using the axiom $A \sqsubseteq \exists P_1$. At the second execution of step 7, the algorithm inserts query q_2, $q(x, s) \leftarrow P_2(x, y) \wedge P_2(\_, y) \wedge \sigma$ using the axiom $\exists P_2^- \sqsubseteq A$. Since the two atoms of the second query unify, $reduce(q, g_1, g_2)$ returns $q(x, s) \leftarrow P_2(x, y) \wedge \sigma$ and since now y is unbound (y does not occur in σ), after application of κ, step 10 inserts the query q_3, $q(x, s) \leftarrow P_2(x, \_) \wedge \sigma$. At the third execution of step 7, the algorithm inserts query q_4, $q(x, s) \leftarrow B(x) \wedge \sigma$ using the axiom $B \sqsubseteq \exists P_2$ and stops.

Note that we need not to evaluate all queries q_i. Indeed, it can easily be verified that for each query q_i and all constants c, the scores of q_3 and q_4 are not lower than all the

other queries q_i and q'. That is, we can restrict the evaluation of the set of reformulated queries to $r(\mathbf{q}, \mathcal{T}) = \{q'', q_3, q_4\}$ only. As a consequence, the top-4 answers to the original query are $[\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle, \langle 3, 0.7\rangle]$, which are the top-4 ranked tuples of the union of the answer sets of q'', q_3 and q_4. □

Computing top-k answers. The main property of the query reformulation algorithm is as follows. It can be shown that

$$ans_k(\mathcal{K}, \mathbf{q}) = \mathtt{Top}_k\{\langle \mathbf{c}, v\rangle \mid q_i \in r(\mathbf{q}, \mathcal{T}), \mathcal{A} \models q_i(\mathbf{c}, v)\}\ .$$

The above property dictates that the set of reformulated queries $q_i \in r(\mathbf{q}, \mathcal{T})$ can be used to find the top-k answers, by evaluating them over the set of instances $\mathcal{A}$ only, without referring to the ontology $\mathcal{T}$ anymore. In the following, we show how to find the top-k answers of the union of the answer sets of conjunctive queries $q_i \in r(\mathbf{q}, \mathcal{T})$.

A naive solution to the top-k retrieval problem is as follows: we compute for all $q_i \in r(\mathbf{q}, \mathcal{T})$ the whole answer set $ans(q_i, \mathcal{A}) = \{\langle \mathbf{c}, v\rangle \mid \mathcal{A} \models q_i(\mathbf{c}, v)\}$, then we compute the union, $\bigcup_{q_i \in r(\mathbf{q}, \mathcal{T})} ans(q_i, \mathcal{A})$, of these answer sets, order it in descending order of the scores and then we take the top-k tuples. We note that each conjunctive query $q_i \in r(\mathbf{q}, \mathcal{T})$ can easily be transformed into an SQL query expressed over $\mathtt{DB}(\mathcal{A})$. The transformation is conceptually simple. The only non-trivial case concerns binary atoms with unbound terms: for any atom in a query $q_i \in r(\mathbf{q}, \mathcal{T})$ of the form $P(\_, x)$, we introduce a view predicate that represents the union of $tab_P[2]$ with $tab_{\exists P^-}$, where $tab_P[2]$ is the projection of tab_P on its second column (the case $P(x, \_)$ is similar). A major drawback of this solution is the fact that there might be too many tuples with non-zero score and hence for any query $q_i \in r(\mathbf{q}, \mathcal{T})$, all these scores should be computed and the tuples should be retrieved. This is *not feasible* in practice, as a there may be millions of tuples in the knowledge base.

A more effective solution consists in relying on existing top-k query answering algorithms for relational databases (see, e.g. [6, 8, 16]), which support efficient evaluations of ranking top-k queries in relational database systems. Though there is no work supporting top-k query answering for disjunctive queries, we can still profitably use top-k query answering methods for relational databases. Indeed, an immediate and much more efficient method to compute $ans_k(\mathcal{K}, \mathbf{q})$ is: we compute for all $q_i \in r(\mathbf{q}, \mathcal{T})$, the top-$k$ answers $ans_k(\mathcal{A}, q_i)$, using e.g. the system RankSQL [16] [6]. If both k and the number, $n_q = |r(\mathbf{q}, \mathcal{T})|$, of reformulated queries is reasonable, then we may compute the union, $U(q, \mathcal{K}) = \bigcup_{q_i \in r(\mathbf{q}, \mathcal{T})} ans_k(\mathcal{A}, q_i)$, of these top-$k$ answer sets, order it in descending order w.r.t. score and then we take the top-k tuples.

As an alternative, we can avoid to compute the whole union $U(q, \mathcal{K})$, so further improving the answering procedure, by relying on a *disjunctive* variant of the so-called *Threshold Algorithm* (TA) [9], which we call *Disjunctive TA* (DTA). We recall that the TA has been developed to compute the top-k answers of a conjunctive query with monotone score combination function. In the following we show that we can use the same principles of the TA to compute the top-k answers of the union of conjunctive queries, i.e. a disjunctive query.

1. First, we compute for all $q_i \in r(\mathbf{q}, \mathcal{T})$, the top-$k$ answers $ans_k(\mathcal{A}, q_i)$, using top-k rank-based relational database engine. Now, let us assume that the tuples in the top-k answer set $ans_k(\mathcal{A}, q_i)$ are sorted in decreasing order with respect to the score.

[6] RankSQL will be available in the middle of this year. Personal communication.

2. Then we process each top-k answer set $ans_k(\mathcal{A}, q_i)$ ($q_i \in r(\mathbf{q}, \mathcal{T})$) in parallel or alternating fashion, and top-down (i.e. the higher scored tuples in $ans_k(\mathcal{A}, q_i)$ are processed before the lower scored tuples in $ans_k(\mathcal{A}, q_i)$).
   (a) For each tuple $\mathbf{c}$ seen, if its score is one of the k highest we have seen, then remember tuple $\mathbf{c}$ and its score $s(\mathbf{c})$ (ties are broken arbitrarily, so that only k tuples and their scores need to be remembered at any time).
   (b) For each answer set $ans_k(\mathcal{A}, q_i)$, let s_i be the score of the last tuple seen in this set. Define the threshold value θ to be $\max(s_1, ..., s_{n_q})$. As soon as at least k tuples have been seen whose score is at least equal to θ, then halt (indeed, any successive retrieved tuple will have score $\leq \theta$).
   (c) Let Y be the set containing the k tuples that have been seen with the highest scores. The output is then the set $\{\langle \mathbf{c}, s(\mathbf{c})\rangle \mid \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{K}, \mathbf{q})$.

The following example illustrates the DTA.

Example 4. Consider Example 3. Suppose we are interested in retrieving the top-3 answers of the disjunctive query $\mathbf{q} = \{q', q''\}$. We have seen that it suffices to find the top-3 answers of the union of the answers to q_3, q_4 and to q''. Let us show how the DTA works. First, we submit q_3, q_4 and q'' to a rank-based relational database engine, to compute the top-3 answers. It can be verified that

$$\begin{aligned} ans_3(\mathcal{A}, q_3) &= [\langle 0, 1.0\rangle, \langle 3, 0.7\rangle, \langle 4, 0.6\rangle] \\ ans_3(\mathcal{A}, q_4) &= [\langle 1, 0.9\rangle, \langle 2, 0.8\rangle, \langle 5, 0.5\rangle] \\ ans_3(\mathcal{A}, q'') &= [\langle 2, 0.84\rangle, \langle 3, 0.64\rangle, \langle 4, 0.36\rangle] \,. \end{aligned}$$

The lists are in descending order w.r.t. the score from left to right. Now we process alternatively $ans_k(\mathcal{A}, q_3)$, then $ans_k(\mathcal{A}, q_4)$ and then $ans_k(\mathcal{A}, q'')$ in decreasing order of the score. The table below summaries the execution of our DTA algorithm. The ranked list column contains the list of tuples processed.

Step	Tuple	s_{q_3}	s_{q_4}	$s_{q''}$	θ	ranked list
1	$\langle 0, 1.0\rangle$	1.0	-	-	1.0	$\langle 0, 1.0\rangle$
2	$\langle 1, 0.9\rangle$	1.0	0.9	-	1.0	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle$
3	$\langle 2, 0.84\rangle$	1.0	0.9	0.84	1.0	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle$
4	$\langle 3, 0.7\rangle$	0.7	0.9	0.84	0.9	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle\langle 3, 0.7\rangle$
5	$\langle 2, 0.8\rangle$	0.7	0.8	0.84	0.84	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle\langle 3, 0.7\rangle$

At step 5 we stop as the ranked list already contains three tuples above the threshold $\theta = 0.84$. So, the final output is

$$ans_k(\mathcal{A}, q_3) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle] \,.$$

Note that not all tuples have been processed. □

As computing the top-k answers of each query $q_i \in r(\mathbf{q}, \mathcal{T})$ requires (sub) linear time w.r.t. the database size (using, e.g. [6]), it is easily verified that the disjunctive TA algorithm is linear in data complexity.

Proposition 3 *Given a DL-Lite KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ and a disjunctive query $\mathbf{q}$ then the DTA computes $ans_k(\mathcal{K}, \mathbf{q})$ in (sub) linear time w.r.t. the size of $\mathcal{A}$.*

Furthermore, the above method has the non-negligible advantage to be based on existing technology for answering top-k queries over relational databases, improves significantly the naive solution to the top-k retrieval problem, and is rather easy to implement.

4 Conclusions

DLs have been proposed as a mean to describe structured knowledge and find a natural application in the context of the Semantic Web. We have presented DL-Lite in which fuzzy predicates are allowed to appear in conjunctive queries. Thus, we may express queries such as "find *cheap* hotels". Such queries are already very common on the Web. To the best of our knowledge, this is the first time this problem has been addressed for classical DLs. A major distinction of DL-Lite is that an answer to a query is a set of tuples *ranked* according to their score. As a consequence, whenever we deal with a huge amount of tuples, the ranking of the answer set becomes the major problem that has to be addressed.

We have shown how to answer disjunctive queries efficiently over a huge set of instances. The main ingredients of our solution is a simple and effective query reformulation procedure, the use of existing top-k query answering technology over relational databases and the DTA algorithm. Indeed, a user query is reformulated into a set of conjunctive queries using the inclusion axioms only and, then, the reformulated queries can be submitted to the top-k query answering engine over a relational database where the tuples have been stored. Finally, the DTA algorithm performs the final computation to retrieve the actual top-k results. We point out that, due to the results described in [5], it is difficult to extend the language proposed here with additional constructs. For instance, it is shown that adding qualified role restrictions $\exists R.C$ would lead to a NLOGSPACE data complexity, which rules out the possibility of using current top-k relational database technology. Furthermore, the complexity result shows that it is the same as for DL-Lite and, thus, whenever DL-Lite can be considered as useful, so is DL-Lite as well.

We note that in [18] we considered the case of top-k query answering within fuzzy DL-Lite. [18] and this work are orthogonal in the sense that in [18] tuples may have a score, but no score combination function is allowed in the query language, while here we consider a classical semantics with score combination function in the query language. The combination of both features is an open direction for further research. Additionally, other topics for future research may be: (i) to verify the applicability of our method to other tractable DLs such as [1]; (ii) to address the problem of top-k query answering to more expressive DLs than DL-Lite; (iii) to improve the DTA by using more sophisticated, but better performing TA-based algorithms such as [10]; and (iv) to improve the core top-k conjunctive query answering technology towards the management of the disjunctive queries.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, pages 364-369, 2005.
2. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
3. Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 16–21, 2003.

4. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005.
5. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proceedings of the 2005 International Workshop on Description Logics (DL-05)*, 2005.
6. Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD Conference*, 2002.
7. CuiMing Chen, Volker Haarslev, and JiaoYue Wang. Las: Extending racer by a large abox store. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL-05)*, 2005.
8. Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, 2002.
9. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.
10. Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kiesling. Optimizing multi-feature queries for image databases. In *The VLDB Journal*, pages 419–428, 2000.
11. Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.
12. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
13. Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of reasoning in very expressive description logics. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 466–471, 2005.
14. Laks V.S. Lakshmanan and Nematollaah Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
15. Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases*. MIT Press, 2001.
16. Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: query algebra and optimization for relational top-k queries. In *SIGMOD Conference*, pages 131–142, 2005.
17. Umberto Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.
18. Umberto Straccia. Answering vague queries in fuzzy dl-lite. In *Proceedings of the 11th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-06)*, 2006.
19. Umberto Straccia. A fuzzy description logic for the semantic web. In Elie Sanchez, editor, *Fuzzy Logic and the Semantic Web*, Capturing Intelligence, chapter 4, pages 73–90. Elsevier, 2006.
20. J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, Potomac, Maryland, 1989.
21. M. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC-82)*, pages 137–146, 1982.
22. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.