Towards Large-Scale Probabilistic OBDA

Joerg Schoenfisch and Heiner Stuckenschmidt

Data- and Web Science Group, University of Mannheim
B6 26, 68159 Mannheim, Germany
{joerg,heiner}@informatik.uni-mannheim.de

Abstract. Ontology-based Data Access has intensively been studied as a very relevant problem in connection with semantic web data. Often it is assumed, that the accessed data behaves like a classical database, i.e. it is known which facts hold for certain. Many Web applications, especially those involving information extraction from text, have to deal with uncertainty about the truth of information. In this paper, we introduce an implementation and a benchmark of such a system on top of relational databases. Furthermore, we propose a novel benchmark for systems handling large probabilistic ontologies. We describe the benchmark design and show its characteristics based on the evaluation of our implementation.

1 Motivation

Ontology-based Data Access (ODBA) has received a lot of attention in the Semantic Web Community. In particular, results on light weight description logics that allow efficient reasoning and query answering provide new possibilities for using ontologies in data access. One approach for ontology-based data access is to rewrite a given query based on the background ontology in such a way that the resulting – more complex – query can directly be executed on a relational database. This is possible for different light-weight ontology languages, in particular the $DL\text{-}Lite$ family [1].

At the same time, it becomes more and more clear that many applications in particular on the (Semantic) Web have to deal with uncertainty in the data. Examples are large-scale information extraction from text or the integration of heterogeneous information sources. To cope with uncertainty, the database community has investigated probabilistic databases where each tuple in the database is associated with a probability indicating the belief in the truth of the respective statement. Querying a probabilistic database requires not only to retrieve tuples that match the query, but also to compute a correct probability for each answer.

The goal of our work is to develop data access methods that can use background knowledge in terms of a light weight ontology and also deal with uncertainty in the data. A promising idea for efficiently computing probabilistic query answers is to use existing approaches for OBDA based on query rewriting and pose the resulting query against a probabilistic database that computes answers with associated probabilities.

A number of approaches have been proposed for combining description logics with probabilistic reasoning. An overview of early approaches is [2], more recent approaches include DISPONTE/BUNDLE [3, 4] or Pronto [5], and Log-linear Description Logics [6]. On the other hand, the logic programming and statistical relational learning community

has developed probabilistic versions of datalog-style languages (e.g. Problog [7]) that can be used to partially model ontological background knowledge. While for many of these languages efficient subsets have been identified (e.g. [4, 5]) and optimized reasoning algorithms have been proposed, none of the existing approaches is really designed to handle large amounts of data as we find on the Web. For example, the full dataset extracted from Web pages by the NELL (Never Ending Language Learning) Project [8] currently contains about 50 million statements with associated probabilities.

Jung et al. have shown that query rewriting for OBDA can directly be lifted to the probabilistic case [9]. Furthermore, they prove that the complexity results and the dichotomy of safe (data complexity in PTIME) and unsafe (#P-hard) queries also carries over. Furthermore, they prove that the complexity results and the dichotomy of safe (data complexity in PTIME) and unsafe (in #P-hard) queries also carries over. To the best of our knowledge, no evaluation on the performance and scalability of the approach was conducted, and there exists no implemented system. We believe, that combining the power of probabilistic database systems with the $DL\text{-}Lite$ approach to ODBA – namely rewriting the query using the background ontology in such a way that the resulting query posed against a database returns the correct results – is a way to scale up to datasets of the size of NELL and beyond.

The main contributions presented in the paper are the following:

- a preliminary implementation of a system that can answer safe probabilistic queries over large probabilistic knowledge bases up to several hundred millions of facts.
- a synthetic benchmark dataset for probabilistic OBDA on the basis of the LUBM benchmark that can be scaled to an arbitrary number of probabilistic statements
- a comparison of the prototype to a state of the art system using that dataset and a real world knowledge base

The paper is structured as follows: We show how the distribution semantics is applied to $DL\text{-}Lite$ in Section 2. Section 3 is concerned with implementing reasoning on top of probabilistic databases. In Section 4 we describe the datasets that can be used for benchmarking probabilistic OBDA. An experimental evaluation of a prototype for large-scale probabilistic OBDA is given in Section 5. Related work is discussed in Section 6 and we conclude and give an outlook in Section 7.

2 $DL\text{-}Lite_R$ and the Distribution Semantics

In this section we briefly introduce $DL\text{-}Lite$, the description logic underlying the OWL 2 QL profile. Then we detail how the distribution semantics for probabilistic description logics [4] is applied to $DL\text{-}Lite_R$.

In $DL\text{-}Lite_R$ concepts and roles are formed in the following syntax [10]:

$$B \rightarrow A \mid \exists R \qquad C \rightarrow B \mid \neg B$$
$$R \rightarrow P \mid P^- \qquad E \rightarrow R \mid \neg R$$

where A denotes an atomic concept, P an atomic role, and P^- the inverse of the atomic role P. B denotes a basic concept, i.e. either an atomic concept or a concept of the form

$\exists R$, where R denotes a basic role, that is, either an atomic role or the inverse of an atomic role. C denotes a general concept, which can be a basic concept or its negation, and E denotes a general role, which can be a basic role or its negation.

A $DL\text{-}Lite_R$ knowledge base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ models a domain in terms of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. A TBox is formed by a finite set of inclusion assertions of the form $B \sqsubseteq C$ or $R \sqsubseteq E$. An ABox is formed by a finite set of membership assertions on atomic concepts and on atomic roles, of the form $A(a)$ or $P(a, b)$ stating respectively that the object denoted by the constant a is an instance of A and that the pair of objects denoted by the pair of constants (a, b) is an instance of the role P.

The semantics of a DL is as an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consisting of a nonempty interpretation domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$:

$$
\begin{aligned}
A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \\
P^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\
(P)^{\mathcal{I}} &= \{(o_2, o_1) | (o_1, o_2) \in P^{\mathcal{I}}\} \\
(\exists R)^{\mathcal{I}} &= \{o \mid \exists o' . (o, o') \in R^{\mathcal{I}}\} \\
(\neg B)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}} \\
(\neg R)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}
\end{aligned}
$$

An interpretation $\mathcal{I}$ is a model of $C_1 \sqsubseteq C_2$, where C_1,C_2 are general concepts if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. Similarly, I is a model of $E_1 \sqsubseteq E_2$, where E_1, E_2 are general roles if $E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}$.

To specify the semantics of membership assertions, the interpretation function is extended to constants, by assigning to each constant a a distinct object $a^{\mathcal{I}} \in \Delta^I$. This enforces the unique name assumption on constants. An interpretation $\mathcal{I}$ is a model of a membership assertion $A(a)$ (resp., $P(a, b)$), if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$).

Given an assertion α and an interpretation $\mathcal{I}$, $\mathcal{I} \models \alpha$ denotes the fact that $\mathcal{I}$ is a model of α. Given a (finite) set of assertions λ, $\mathcal{I} \models \lambda$ denotes the fact that $\mathcal{I}$ is a model of every assertion in λ. A model of a KB $K = \langle T, A \rangle$ is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models T$ and $\mathcal{I} \models A$. A KB is satisfiable if it has at least one model. A KB K logically implies an assertion α, written $K \models \alpha$, if all models of K are also models of α. Similarly, a TBox T logically implies an assertion α, written $T \models \alpha$, if all models of T are also models of α.

To account for uncertainty, the distribution – or possible world – semantics is widely used in logic programming. The basic idea is to allow the annotation of axioms in a knowledge base with probabilities, under the assumption that all axioms are mutually independent. Riguzzi et al. applied the semantics to description logics [4]. They presented a prototypical application to the DL $\mathcal{SHOIN}(\mathbf{D})$ and gave an approach for performing inference. Transferring the semantics to $DL\text{-}Lite_R$ is straightforward.

Following the syntax and notation of Riguzzi et al., a probabilistic $DL\text{-}Lite_R$ knowledge base $\mathcal{K}$ is a set of certain axioms $\mathcal{C}$ and probabilistic axioms $\mathcal{E}$. Certain axioms have the form of regular DL axioms. Probabilistic axioms are in the form $p :: E$ where p is a real-valued probability in (0,1), and E is a DL axiom. The probability p is interpreted as

epistemic probability, i.e. as a degree of belief or subjective probability, not a statistical or objective probability.

Each probabilistic axiom is associated with a Boolean random variable. Different possible worlds are then obtained by assigning values to every random variable. A possible world w is a set of axioms containing all certain axioms and all axioms whose Boolean variable has the value 1.

Again following [4], the semantics are formalized as follows: A tuple (E_i, k) is called *atomic choice*. E_i is the i-th probabilistic axiom and $k \in \{0, 1\}$ indicates if E_i is included in the current world. All choice are assumed to be mutually independent. For a set κ of atomic choices to be consistent, there may only exist one choice for each probabilistic axiom. If κ is consistent it is also called *composite choice*. A *selection* σ is a total composite choice, i.e. it contains an atomic choice (E_i, k) for every $E_i \in \mathcal{K}$. A selection σ identifies a possible world w_σ in this way: $w_\sigma = \mathcal{C} \cup \{E_i | (E_i, 1) \in \sigma\}$. $\mathcal{W}_\mathcal{K}$ is the set of all possible worlds. The probability of world w_σ is $P(w_\sigma) = \prod_{(E_i,1)\in\sigma} p_i \prod_{(E_i,0)\in\sigma} 1 - p_i$, where p_i is the probability for axiom E_i. $P(w_\sigma)$ is a probability distribution over worlds, i.e. $\sum_{w\in\mathcal{W}_\mathcal{K}} P(w) = 1$.

In general, the distribution semantics allows probabilistic axioms in the TBox and the ABox. However, in this paper we restrict the uncertainty to the ABox to facilitate the rewriting step and the aggregation of the final result. Only modeling uncertainty in the ABox is common in knowledge extraction frameworks for the web like NELL, and thus has largely no impact on the general applicability of our approach. During the rest of this paper we call the probabilistic description logic defined above *ProbQL*.

3 Implementing Reasoning on Top of Probabilistic Databases

In this section, we first briefly recall the idea of first-order rewritability of queries in $DL\text{-}Lite$ and then show that the query rewriting approach proposed by [10] can be used on top of tuple independent probabilistic databases for answering queries in *ProbQL* without changing the semantics of answers.

3.1 Query Rewriting

Query processing in $DL\text{-}Lite_R$ is based on the idea of first-order reducibility. This means that for every conjunctive query q we can find a query q' that produces the same answers as q by just looking at the A-Box. Calvanese et al. also define a rewriting algorithm that computes a q' for every q by applying transformations that depend on the T-Box axioms.

Given a consistent T-Box $\mathcal{T}$ the algorithm takes a conjunctive query q_0 and expands it into a union of conjunctive queries U starting with $U = \{q_0\}$. The algorithm successively extends U by applying the following rule:

$$U = U \cup \{q[l/r(l, I)]\}$$

where q is a query in U, l is a literal in q, I is an inclusion axiom from $\mathcal{T}$ and r is a replacement function that is defined as follows:

$\mathbf{l}$	$\mathbf{I}$	$\mathbf{r}(l, I)$
$A(x)$	$A' \sqsubseteq A$	$A'(x)$
$A(x)$	$\exists P \sqsubseteq A$	$P(x, \_)$
$A(x)$	$\exists P^- \sqsubseteq A$	$P(\_, x)$
$P(x, \_)$	$A \sqsubseteq \exists P$	$A(x)$
$P(x, \_)$	$\exists P' \sqsubseteq \exists P$	$P'(x, \_)$
$P(x, \_)$	$\exists P'^- \sqsubseteq \exists P$	$P'(\_, x)$

$\mathbf{l}$	$\mathbf{I}$	$\mathbf{r}(l, I)$
$P(\_, x)$	$A \sqsubseteq \exists P^-$	$A(x)$
$P(\_, x)$	$\exists P' \sqsubseteq P^-$	$P'(x, \_)$
$P(\_, x)$	$\exists P'^- \sqsubseteq \exists P^-$	$P'(\_, x)$
$P(x, y)$	$P' \sqsubseteq P$	$P'(x, y)$
$P(x, y)$	$P'^- \sqsubseteq P^-$	$P'(x, y)$
$P(x, y)$	$P' \sqsubseteq P^-$	$P'(y, x)$
$P(x, y)$	$P'^- \sqsubseteq P$	$P'(y, x)$

Here '_' denotes an unbound variable, i.e., a variable that does not occur in any other literal of any of the queries.

3.2 Correctness of Query Processing

Implementing *ProbQL* on top of probabilistic databases can now be done in the following way. The A-Box is stored in the probabilistic database, the query is rewritten and posed against the database. It can easily be shown that the idea that a rewritten query has the same probability given a knowledge base with empty T-Box as the original query given a complete knowledge base. Further, the semantics of queries over a *ProbQL* A-Box with empty T-Box directly corresponds to the tuple-independence semantics used in probabilistic databases [11], thus queries posed to correctly constructed probabilistic database have the same probability as a *ProbQL* query with empty T-Box. Results on first-order rewritabnility of DL-lite we get, that $P(Q|\mathcal{KB})$ does not change as $\mathcal{T}, \mathcal{A} \models Q$ if and only if $\mathcal{A} \models Q'$. Further, as $P(\mathcal{KB}|\mathcal{TKB})$ only depends on $\mathcal{A}$ this part also stays unchanged.

3.3 Safe Queries and Query Rewriting

Over the past decade, the database community has developed efficient methods for querying uncertain information in probabilistic databases. Important results are the introduction of the independent tuple model for probabilistic data as well as a complete characterization of queries that can be computed in polynomial time. We briefly review these results in this section.

Extensional Query Processing It has been shown that the key to efficient query processing in probabilistic databases is to avoid the computation of complex event descriptions and to directly compute the probability of a complex query from the probabilities of subqueries. This approach, referred to as extensional query processing, has been shown to correctly compute the probability of queries for the class of tractable queries using the following recursive algorithm:

Algorithm 1 Extensionally compute P(Q) (from [12])

Require: A conjunctive query Q in CNF, a tuple independent probabilistic database
1: Q is a conjunctive query in CNF: Compute symbol components $Q = Q_1 \wedge \cdots \wedge Q_m$
2: **if** $m \geq 2$ **then**
3: **return** $\prod_{i=1,\cdots,m} P(Q_i)$
4: **end if**
5:
6: $Q = Q_1 \wedge \cdots \wedge Q_k$ is a symbol-connected query in CNF:
7: **if** $k \geq 2$ **then**
8: **return** $-\sum_{s\subseteq[n],s\neq\emptyset}(-1)^{|s|}P\left(\bigvee_{i\in s} Q_i\right)$
9: **end if**
10:
11: Q is a disjunctive query: Compute symbol components $Q = Q_1 \vee \cdots \vee Q_m$
12: **if** $m \geq 2$ **then**
13: **return** $1 - \left(\prod_{i=1,\cdots,n} 1 - P(Q_i)\right)$
14: **end if**
15:
16: $Q = Q_1 \vee \cdots \vee Q_k$ is a symbol-connected disjunctive query:
17: **if** $Q = t$ has no variables **then**
18: **return** $P(t)$
19: **end if**
20:
21: **if** Q has a separator variable z **then**
22: **return** $1 - \left(\prod_{a\in ADom} 1 - P(Q[a/x])\right)$
23: **else**
24: **return false**
25: **end if**

Symbol components are defined as follows [11]: Let $Q = d_1 \wedge \ldots \wedge d_k$ be a UCQ in CNF, and $K_1, \ldots, K_m$ the connected components for the set of queries $\{d_1, \ldots, d_k\}$. The symbol-components of Q are $Q1 = \bigwedge_{i\in K_1} d_i$, …, $Q_m = \bigwedge_{i\in K_m} d_i$. Then the probability $P(Q) = P(Q_1) \cdot \ldots \cdot P(Q_m)$. If $m = 1$, then Q is *symbol-connected*.

Each recursion of the algorithm processes a simpler subexpression, until the probability of an individual can be read directly from the database. Queries that can be completely processed using the steps above are called *safe*. It has been shown that safe queries exactly correspond to queries that can be computed in PTIME whereas queries that cannot be completely processed using these steps – in particular queries for which we cannot find a separator variable in line 19 – are in #P-hard. This means that we can use the notion of safeness and the processing steps above to analyze the general complexity of certain classes of queries.

3.4 Mapping Safe Queries to SQL

The mapping of safe conjunctive queries to SQL has two main parts: 1. Translating single atoms and queries to SELECT statements. 2. Extensional query processing to calculate the probability of each answer (row in the SQL result).

Translating the CQs to SQL is basically done by creating a sub-select for each atom in the query and joining them on shared variables. This translation step is already part of existing non-probabilistic OBDA systems that use query rewriting. The probabilistic reasoning additionally needs four functions to calculate probabilities:

- Independent Join (line 3 of Algorithm 1): Calculating the probability of an independent join is implemented in the SELECT of the query. The (aggregated) probabilities of the joined tables or sub-selects are multiplied for each result row.
- Möbius Inversion Formula (line 8): Similarly, the Möbius Inversion Formula is also computed inside the SELECT. To handle the changing signs and subsets of the query, it has to be wrapped in another sub-select.
- Independent Union (line 13): The independent union is implemented as a custom aggregate in the database. The aggregate function multiplies the inverse probability of each row and returns the inverse of that product.
- Independent Project (line 22): The independent project is implemented analogously to the independent join, but using the inverse probabilities.

Independent Join, Möbius Inversion Formula and Independent Project process elements of a single result row, whereas Independent Union aggregates the final probabilities of multiple rows.

4 Benchmark Data for Probabilistic OBDA

We use two different datasets for our experimental evaluation. The first dataset is the ontology and knowledge base created by NELL, which presents a large real-world dataset consisting of uncertain data. Second, to assess the scalability of the approach, we created a modified version of the Lehigh University Benchmark (LUBM), a synthetic benchmark for OWL reasoners that generates datasets of various sizes.

4.1 NELL

NELL is an Open Information extraction system that extracts facts from text found in a large corpus of web pages. As a result, NELL generates triples like `wifeof(katie_holmes, cruise)`, called candidate beliefs, that are annotated with different levels of confidence in terms of a number in the range $(0, 1]$.

Within the context of our approach these candidate beliefs form the A-Box of our *ProbQL* knowledge base, while the confidences are interpreted as probabilities. NELL organizes extracted facts in a terminology consisting of concepts (called categories in NELL context) and roles (relations) and specifies domain and range restrictions, property symmetry, and disjointness of concepts and properties. We use the $DL\text{-}Lite_R$ fragment of this terminology as T-Box of our *ProbQL* knowledge base. We use the high confidence knowledge base of NELL (iteration 860) which contains only facts with a score of at least 0.75. It contains 2.3 million extracted facts about 1.8 million objects as compared to the full dataset with about 50 million. The T-Box, which is the same for all datasets, consists of 558 concepts, 1 255 properties, and 5 132 axioms.

Dataset	Assertions
NELL full	2 259 750
NELL filtered	467 943

Dataset	Assertions	% distinct
LUBM 1	717 250	54.77
LUBM 10	7 232 663	55.69
LUBM 100	71 698 666	55.66
LUBM 200	143 311 100	55.67
LUBM 500	361 432 844	55.12
LUBM 1000	719 097 512	55.32

Table 1. Size of the different NELL and LUBM datasets.

To show the benefits and the scalability of our approach, we defined the following queries that are posed against the *ProbQL* version of NELL.

$Q_A(X) \Leftarrow person(X)$
$Q_B(X) \Leftarrow person(X), bornin(X, paris)$
$Q_C(X) \Leftarrow book(X), movie(X)$
$Q_D(Z) \Leftarrow hasParent(X, Y), hasparent(Y, Z)$
$Q_E(X) \Leftarrow actor(X), directordirectedmovie(X, Y), writerwrotebook(X, Z)$
$Q_F(X) \Leftarrow politician(X), actor(X), hasoffice(X, president)$

We use this dataset mainly to investigate the benefits of using background knowledge and reasoning on top of probabilistic data in terms of increased recall.

4.2 Probabilistic LUBM

The Lehigh University Benchmark (LUBM) [13] is a well known and widely used benchmark for OWL-based reasoning systems. Lutz et al. published a $DL\text{-}Lite_R$ version of LUBM [14]. Additionally, to restricting the expressivity of the ontology, they modified it to make it more suitable in an OBDA setting: First, they added multiple concept inclusions with existential restriction on the right hand side, second they extended the class hierarchy to be closer to real-world ontologies in its size.

We chose to extend LUBM over other synthetic benchmarks like SP^{2}Bench [15], BSBM [16], or FishMark [17]. SP^{2}Bench and BSBM only provide a very simple or no ontology, but rather focus on complex queries. FishMark contains an expressive ontology more suitable for our evaluation. However, it does not provide a generator for datasets of various sizes. Lutz et al.'s version of the LUBM ontology is of sufficient complexity to evaluate the scalability of reasoning in an OBDA setting, and it offers the possibility to generate datasets of different sizes.

We generated the benchmark dataset for probabilistic OBDA in two steps: 1. We extended the generator to create probabilistic ABoxes. 2. To increase the complexity of the probabilistic reasoning, we created redundancies in the dataset. In the first step we extended the implementation of the data generator to attach probabilities to every ABox axiom. Those probabilities are randomly distributed in (0,1]. We did not include a fixed percentage of certain axioms. The generator thus creates datasets of various

size with probabilistic axioms. However, each axiom is contained exactly once, thereby trivializing the calculation of the final probabilities in a result set. To alleviate this, we used the option to change the `seed` that LUBM uses to determine the number of instances of departments, professors, students, etc. For every dataset size, we generated five ABoxes each with a different seed (0, 1, 42, 776, 141984). Combined, these five ABox serve as one probabilistic benchmark dataset. Note that our probabilistic version of LUBM is roughly five times larger than the normal LUBM of the same size, i.e. LUBM 1 contains only one university, whereas probabilistic LUBM 1 contains five different versions of that university, with different numbers of departments, professors, students, etc.

In the evaluation we used the original LUBM queries for which probabilities can be computed efficiently, i.e. queries 1, 3–6, and 10–14:

$$
\begin{aligned}
Q_1(X) &\Leftarrow takesCourse(X, univ0\_dept0), type(X, graduateStudent)\\
Q_3(X, Y_1, Y_2, Y_3) &\Leftarrow publicationAuthor(X, univ0\_asstProf0), type(X, publication)\\
Q_4(X) &\Leftarrow worksFor(X, univ0\_dept0), name(X, Y_1), emailAddress(X, Y_2),\\
&\qquad telephone(X, Y_3), type(X, professor)\\
Q_5(X) &\Leftarrow memberOf(X, univ0\_dept0), type(X, person)\\
Q_6(Z) &\Leftarrow type(X, student)\\
Q_{10}(X) &\Leftarrow takesCourse(X, univ0\_graduateCourse0), type(X, student)\\
Q_{11}(X) &\Leftarrow subOrgOf(X, univ0), type(X, researchGroup)\\
Q_{12}(X, Y) &\Leftarrow worksFor(X, Y), type(X, chair), subOrgOf(Y, univ0),\\
&\qquad type(Y, department)\\
Q_{13}(X) &\Leftarrow hasAlumnus(univ0, X), type(X, person)\\
Q_{14}(X) &\Leftarrow type(X, undergraduateStudent)
\end{aligned}
$$

Q_{11} and Q_{12} require the reasoner to handle the transitive object property `subOrgOf`. However, transitive object properties are not allowed in $DL\text{-}Lite_R$. To circumvent this and still be able to use this query in the evaluation, we manually extended those queries to handle transitivity up to the maximum depth occurring in the data (in this case 2).

5 Experimental Evaluation

We evaluate our implementation of query processing for *ProbQL* on the two datasets presented in the previous section. Our main goal is to show that our implementation scales to vers large A-Boxes and outperforms existing methods on safe queries.

5.1 Setting

Within our experiments we focus on answering the following two questions:

1. What are the benefits of exploiting the TBox by using it in the query rewriting process for a dataset like NELL?
2. How well does our algorithm scale with respect to different types of queries and subsets of NELL and a probabilistic LUBM, and compared to another system?

For answering the first question, we compare query results with and without query rewriting. We expect that rewriting the queries yields larger result sets. In particular, we expect that many interesting results are missed out when we ask the query directly without any expansion.

For answering the second question, we compare our implementation against the ProbLog system [7], which also uses the independent tuple semantics. While ProbLog does not support the complete expressivity of $DL\text{-}Lite$[1], it returns identical results for any safe conjunctive query over the dataset. For the comparison with ProbLog, we used a subset of the data consisting of about half a million facts. Additionally, to assess the general scalability of the approach, we run our implementation on different sizes of probabilistic LUBM, i.e. 1, 10, 100, 200, 500, and 1000 universities.

Experiments were run on a virtual machine with 4 cores (2.4GHz) and 16GB RAM running Ubuntu 14.10 Server. We used PostgreSQL 9.4 64bit and ProbLog 2. We used to default settings of the database and did no special tuning apart from increasing the available RAM. The NELL dataset was loaded into a single table. The LUBM datasets use different tables for class, object, and data property assertions, one of each for different sizes of the benchmark. Query rewriting was done manually at this point, but we do not expect a significant impact on this step on the overall performance. As ProbLog always has to load all the data and does not provide persistent storage like a database system, we measured the time ProbLog takes to parse the file without a query and subtracted that amount from the query time.

An existing probabilistic database like MayBMS was not used, as we encountered serious issues with complex JOINs resulting from the query rewriting.

5.2 Results

NELL Dataset Table 2 shows the results of comparing query answering with and without rewriting.

	Plain		Rewritten	
	# res.	# pred.	# res.	# pred.
Q_A	5 405	1	319 986	148
Q_B	1	2	4	152
Q_C	352	2	414	12
Q_D	0	2	80	40
Q_E	0	3	1	11
Q_F	2	4	14	29

Table 2. Number of results with and without reasoning, and increase in query size (predicates)

The number of results generally increases – sometimes dramatically (cf. Q_A) – and we can even find answers to Q_D which produced no results without rewriting. The large increase in results for Q_A is due to *person* being a very general concept of the NELL

[1] In particular, axioms of the form $A \sqsubseteq \exists R$ cannot be represented in ProbLog.

hierarchy and most instances are described using more specific concepts. Q_D has no answers without rewriting because the relation *personhasparent* is never used, but only its inverse *parentofperson*.

Exploiting the T-Box often changes the probabilities for an individual answer to a query as new evidence is added to the computation. For example the probability for *concept:person:sandy* being a *person* in Q_A or Q_B increase from 0.96875 to 1.0. The knowledge base only states that Sandy is a *person* with probability 0.96875. Through the rewriting step, the statement that Sandy graduated from State University with probability 1.0 is also included, resulting in her definitely being a person because of *graduatedfrom* having *person* as domain.

Tables 3 and 4 show the results of comparing *ProbQL* with ProbLog.

	full	filtered
ProbLog	24.393 sec	5.383 sec
SQL Loading	193.040 sec	12.163 sec
SQL Indexing	1 007.291 sec	47.427 sec

Table 3. Dataset loading times (sec)

As expected our approach takes significantly more time loading the data as index structures have to be created and ProbLog only seems to do minimal preprocessing. The results in Table 4 show, however, that this effort is overcompensated by more efficient query answering.

	Q_A	Q_B	Q_C	Q_D	Q_E	Q_F
ProbLog (filtered)	-	97.667	1.812	-	2.673	9.589
ProbLog (full)	-	-	-	-	-	-
Prob. SQL (filtered)	10.423	3.524	0.107	0.024	1.421	0.628
Prob. SQL (full)	8.846	5.488	0.097	0.011	0.888	0.617
SQL (full)	5.002	3.196	0.017	0.009	0.637	0.340

Table 4. Query performance in seconds, averaged over 10 runs

Table 4 shows the time needed for answering queries over the full dataset and the reduced one. To accommodate for the fact that ProbLog always has to load the data anew, loading time has been subtracted from the query times for ProbLog shown in Table 4.

The query response times clearly show that our database-driven approach is more efficient for handling large datasets. ProbLog is not able to answer any of the questions using the full dataset within a 30 minute timeout. Also for the filtered dataset, ProbLog fails for Q_A and Q_D with an out of memory error and not with a timeout as for the full dataset. For the queries where both return answers, our approach is between 15 and 30 times faster. We can observe that query processing even becomes more efficient for

the larger dataset. After analyzing the generated query plans, we found that the query planner chooses a suboptimal query plan for the smaller dataset. We suppose this is due to weaker statistics. The overhead of the probabilistic SQL compared to the plain rewritten SQL seems to be proportional to the number of computed answers. Q_A, Q_C and Q_D, and Q_F, with a larger number of results, are twice to five times as slow as the plain queries; Q_B and Q_E with very few results show almost no difference in query time.

LUBM Dataset Table 5 shows the results using the probabilistic LUBM datasets. We only compared the performance of our implementation on different size of the data. We ran the queries with a timeout of 60 minutes. ProbLog is not able to handle even the smallest of those datasets.

	Q1	Q3	Q4	Q5	Q6	Q10	Q11	Q12	Q13	Q14
1	< 0.1	< 0.1	1.5	2.7	0.7	0.3	0.2	0.8	0.3	0.3
10	< 0.1	1.0	3.4	27.3	7.4	2.4	0.2	0.9	2.9	2.4
100	< 0.1	32.2	50.5	350.9	74.2	33.2	0.3	2.0	76.2	33.2
200	< 0.1	100.0	134.6	2 622.2	172.1	63.9	0.5	3.2	172.5	63.9
500	< 0.1	201.8	440.2	-	508.0	192.0	1.1	6.9	612.1	1 941.9
1000	< 0.1	643.8	904.7	-	874.7	365.1	1.6	232.3	1 430.3	-

Table 5. Query response times (seconds) of our implementation on various sizes of the probabilistic LUBM dataset. A timeout (response time $>$ 60 minutes) is denoted as "-".

The query response times show, that in general, the probabilistic reasoning does not have a negative impact on scalability. Overall, the times increase linearly in the size of the data. Query 1, which has a constant result that does not change with the size of the dataset, also has a constant response time. When processing Query 5, the database erroneously scans the complete table of data property assertions, which takes most of the time for computing results. This could be alleviated by tuning the query planner, resulting in a better query execution plan. Query 13 and especially Query 14 produce a large number of results, thus they become I/O-bound for larger datasets, i.e. their performance is limited by disk speed, resulting in a large jump in the query time. The disks for our test virtual machine are attached via network, resulting in this large drop in performance.

Dataset	LUBM 1	LUBM 10	LUBM 100	LUBM 200
QMpH	418.6	66.4	5.4	0.2
Optimum	514.3	75.0	5.8	1.0
%	81.4%	88.5%	93.1%	20%

Table 6. Query mixes per hour (QMpH) for different dataset sizes. The number indicates how often the set of benchmark queries could be executed within one hour. The queries are executed in random order.

To evaluate the scalability under a more realistic workload we tested how often the set of all ten queries can be executed within one hour (inspired by the BSBM benchmark). The queries are executed in random order. Table 6 shows the number of query mixes processed in an hour for different sizes of the LUBM dataset. Up to 70 million facts, the performance scales well and is close to the expected optimum based on the query times under ideal settings. For the smaller datasets, the response time are slightly farther away from the optimum due to a relatively higher overhead of establishing a database connection etc. Beyond 70 million facts, there is a huge drop in performance due to I/O-bound queries and the poor I/O performance of the virtual machine.

6 Related Work

Apart from ProbLog, two other systems for probabilistic reasoning with a similar semantic are Pronto [5] and BUNDLE [3]. They can handle probabilistic knowledge bases formulated in $\mathcal{SROIQ}$ and $\mathcal{SHOIN}$(D), respectively. However, their main focus is not pOBDA, but probabilistic TBox reasoning (classification, satisfiability, ...), thus their performance in query answering is very limited. Both can only run simple instance checking for single individuals and classes. Probabilistic deductive databases [18] provide a similar solution, but to the best of our knowledge there is no system available and thus it is hard to estimate their scalability to large-scale knowledge bases.

Regarding the benchmark dataset, Klinov et al. [19] proposed a systematic approach to evaluate reasoning in probabilistic description logics which is, however, more geared towards complex TBoxes and not large-scale query answering. Lanti et al. [20] very recently published a dataset, based on real world data, specially tailored for benchmarking OBDA systems. They also provide a generator to scale the dataset in size. It will be interesting to analyze their dataset and also extend it for benchmarking probabilistic OBDA systems.

7 Conclusion and Future Work

In this paper we described a preliminary implementation of a probabilistic OBDA system for large-scale knowledge bases. It combines tractability for a certain class of queries with the benefits of ontology-based query rewriting. While making many simplifying assumptions the approach is well suited for large-scale knowledge bases with facts generated using machine learning techniques and provides a pragmatic alternative for theoretically more interesting but less feasible models as the one proposed in [3, 5].

We used the NELL knowledge base as a real-world example and a probabilistic extension of LUBM to evaluate the system. We demonstrated that it scales well compared to another state-of-the-art system and is able to compute query answers in a reasonable amount of time for knowledge bases containing several million facts.

We plan to provide a stable implementation of the approach in the near future and address the problem of uncertain T-Box elements. Furthermore, we plan to develope a more thorough benchmark based on recent proposals for benchmarks specifically aimed at OBDA [20].

Acknowledgement

The authors want to thank Christian Meilicke for his ongoing support and fruitful discussions about the topic of this paper.

References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. Artif. Intell. Res. **36** (2009) 1–69
2. Lukasiewicz, T., Straccia, U.: Managing uncertainty and vagueness in description logics for the Semantic Web. Web Semant. **6**(4) (2008) 291–308
3. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: BUNDLE: A reasoner for probabilistic ontologies. Web Reason. Rule Syst. (2013) 183—-197
4. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Probabilistic Description Logics under the Distribution Semantics. Semant. Web J. (2014)
5. Klinov, P., Parsia, B.: Pronto: A Practical Probabilistic Description Logic Reasoner. Lect. Notes Comput. Sci. **7123** (2013) 59–79
6. Niepert, M., Noessner, J., Stuckenschmidt, H.: Log-linear description logics. In: IJCAI Int. Jt. Conf. Artif. Intell. (2011) 2153–2158
7. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: IJCAI Int. Jt. Conf. Artif. Intell. (2007) 2468–2473
8. Carlson, A., Betteridge, J., Kisiel, B.: Toward an Architecture for Never-Ending Language Learning. Proc. Conf. Artif. Intell. (2010) 1306–1313
9. Jung, J.C., Lutz, C.: Ontology-Based Access to Probabilistic Data with OWL QL. In: Semant. Web - ISWC 2012, Springer (2012) 182—-197
10. Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. J. Autom. Reason. **39**(3) (July 2007) 385–429
11. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool Publishers (2011)
12. Dalvi, N., Suciu, D.: The dichotomy of probabilistic inference for unions of conjunctive queries. J. ACM **59**(6) (2012) 1–87
13. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. In: Web Semant. Volume 3. (2005) 158–182
14. Lutz, C., Seylan, I., Toman, D., Wolter, F.: The combined approach to OBDA: Taming role hierarchies using filters. Lect. Notes Comput. Sci. **8218 LNCS** (2013) 314–330
15. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^2 Bench: A SPARQL Performance Benchmark. Data Eng. **25** (2009) 222–233
16. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. Int. J. Semant. Web Inf. Syst. **5**(2) (2001) 1–24
17. Bail, S., Alkiviadous, S., Parsia, B., Workman, D., Van Harmelen, M., Goncalves, R.S., Garilao, C.: FishMark: A linked data application benchmark. CEUR Workshop Proc. **943** (2012)
18. Lakshmanan, L.V.S., Sadri, F.: Probabilistic Deductive Databases. SLP (June 1994) 254–268
19. Klinov, P., Parsia, B.: Optimization and Evaluation of Reasoning in Probabilistic Description Logic:Towards a Systematic Approach. (D) (2008) 213–228
20. Lanti, D., Rezk, M., Xiao, G., Calvanese, D.: The NPD Benchmark : Reality Check for OBDA Systems. Proc. 18th Int. Conf. Extending Database Technol. (2015)