Learning Probabilistic Description Logics
Theories

Riccardo Zese
Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
riccardo.zese@unife.it

Abstract. Uncertain information is ubiquitous in real world domains and in the Semantic Web. Recently, the problem of representing this uncertainty in description logics has received an increasing attention. In probabilistic Description Logics, knowledge bases contain numeric parameters that are often difficult to specify for a human. Moreover, the information are incomplete and poorly structured. On the other hand, data is usually available about the domain that can be leveraged for tuning the parameters and learn the structure of the information. In this paper we consider the problem of learning both the structure and the parameters of Probabilistic Description Logics under the DISPONTE semantics. We overview two systems we have implemented: EDGE, that returns the value of the probabilities associated with axioms tuned using an Expectation Maximization algorithm, and LEAP, that exploits EDGE and the system CELOE to learn both the structure and the parameters of DISPONTE knowledge bases.

1 Introduction
In the last few years, the problem of representing uncertainty in description logics (DLs for short) has received an increasing attention due to the ubiquity of uncertain information in real world domains. DLs are at the basis of the Web Ontology Language (OWL for short), a family of knowledge representation formalisms used for modeling information of the Semantic Web [11]. Various researchers have presented proposals for allowing DLs to represent uncertainty [18, 33, 8, 17, 32]. In [26] we presented DISPONTE, a probabilistic semantics for DLs based on the distribution semantics that allows probabilistic assertional and terminological knowledge.

In order to allow inference over the information in the Semantic Web, many efficient DL reasoners, such as Pellet [31], RacerPro [9] and HermiT [30], have been developed. Despite the availability of many DL reasoners, the number of probabilistic reasoners is quite small. In [20, 24] we presented BUNDLE, a reasoner based on Pellet that extends it by allowing to perform inference on DISPONTE theories. We have also implemented two reasoners, TRILL [35] and TRILLP [36, 34], that are written in Prolog. They exploit the backtracking system of the language for managing the non-deterministic operators used in the inference process.
For allowing the computation of the probability of queries, we need Knowledge Bases (KBs for short) containing meaningful parameters associated to the probabilistic axioms. One of the main problems is that specifying these values of probability is a difficult task for humans. However, we can leverage the data available about the domain for tuning the parameters. Furthermore, in some cases the KBs contain information that are poorly structured and incomplete.

In this paper we present a machine learning approach for learning the parameters of probabilistic ontologies from data and a second approach for learning both the structure and the parameters. The first algorithm, called EDGE for “Em over bDds for description logics paramEter learning” [22, 23], starts from examples of instances and non-instances of concepts and calls BUNDLE for building the Binary Decision Diagrams (BDDs) that represent their explanations from the theory. The parameters are then tuned using an Expectation Maximization algorithm [7] over the BDDs in an efficient way.

The second algorithm, called LEAP for “LEArning Probabilistic description logics” [25], combines the learning system CELOE, used to build new (equivalence and subsumption) axioms that can be added to the KB, with EDGE, used to learn the parameters of these probabilistic axioms.

The paper is organised as follows. Section 2 briefly introduces SHOIN($\mathcal{D}$) and presents the DISPONTE semantics. Section 3 briefly introduce the inference algorithms we have developed and in particular BUNDLE, that is used in EDGE. Section 4 presents the EDGE system while section 5 presents the LEAP algorithm. Section 6 discusses related work and section 7 discusses our future plans. Finally, Section 8 concludes the paper.

2 Description Logics and the DISPONTE Semantics

DLs are knowledge representation formalisms represented using a syntax based on concepts, basically sets of individuals of the domain, and roles, sets of pairs of individuals of the domain. In this section, we recall the expressive description logic SHOIN($\mathcal{D}$) [17], that is at the basis of OWL DL.

Let $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{I}$ be sets of atomic concepts, roles and individuals. A role is either an atomic role $R \in \mathcal{R}$ or the inverse $R^-$ of an atomic role $R \in \mathcal{R}$. We use $\mathcal{R}^-$ to denote the set of all inverses of roles in $\mathcal{R}$. An RBox $\mathcal{R}$ consists of a finite set of transitivity axioms $\text{Trans}(R)$, where $R \in \mathcal{R}$, and role inclusion axioms $R \sqsubseteq S$, where $R, S \in \mathcal{R} \cup \mathcal{R}^-$.

Concepts are defined by induction as follows. Each $C \in \mathcal{A}$ is a concept, $\bot$ and $\top$ are concepts, and if $a \in \mathcal{I}$, then $\{a\}$ is a concept. If $C_1$ and $C_2$ are concepts and $R \in \mathcal{R} \cup \mathcal{R}^-$, then $(C_1 \cap C_2)$, $(C_1 \sqcup C_2)$, and $\neg C$ are concepts, as well as $\exists R.C$, $\forall R.C$, $\geq nR$ and $\leq nR$ for an integer $n \geq 0$. A TBox $\mathcal{T}$ is a finite set of concept inclusion axioms $C \sqsubseteq D$, where $C$ and $D$ are concepts. We use $C \equiv D$ to abbreviate $C \sqsubseteq D$ and $D \sqsubseteq C$. An ABox $\mathcal{A}$ is a finite set of concept membership axioms $a : C$, role membership axioms $(a,b) : R$, equality axioms $a = b$ and inequality axioms $a \neq b$, where $C$ is a concept, $R \in \mathcal{R}$ and $a,b \in \mathcal{I}$. 

2
A knowledge base $K = (T, R, A)$ consists of a TBox $T$, an RBox $R$ and an ABox $A$. A knowledge base $K$ is usually assigned a semantics in terms of set-theoretic interpretations $I = (\Delta^T, \mathcal{I})$, where $\Delta^T$ is a non-empty domain and $\mathcal{I}$ is the interpretation function that assigns an element in $\Delta^T$ to each $a \in I$, a subset of $\Delta^T$ to each $C \in A$ and a subset of $\Delta^T \times \Delta^T$ to each $R \in R$.

$S\text{HOIN}(D)$ adds to $S\text{HOIN}$ datatype roles, i.e., roles that map an individual to an element of a datatype such as integers, floats, etc. Then new concept definitions involving datatype roles are added that mirror those involving roles introduced above. We also assume that we have predicates over the datatypes.

A query $Q$ over a KB $K$ is an axiom for which we want to test the entailment from the knowledge base, written $K \models Q$. The entailment test may be reduced to checking the unsatisfiability of a concept in the knowledge base, i.e., the emptiness of the concept. For example, the entailment of the axiom $C \subseteq D$ may be tested by checking the satisfiability of the concept $C \cap \neg D$.

DISPONTE [26] applies Sato’s distribution semantics [28] of probabilistic logic programming to DLs. Under this semantics, a logic program defines a probability distribution over normal logic programs (worlds). Then the distribution is extended to a joint distribution of the query and the programs from which the probability of the query is obtained by marginalization.

In DISPONTE, a probabilistic knowledge base $K$ is a set of certain axioms or probabilistic axioms in which each axiom is independent from the others. Certain axioms take the form of regular DL axioms while probabilistic axioms are $p :: E$ where $p$ is a real number in $[0,1]$ and $E$ is a DL axiom. In DISPONTE, the probability $p$ can be interpreted as an epistemic probability, i.e., as the degree of our belief in axiom $E$. For example, a probabilistic concept membership axiom $p :: a : C$ means that we have degree of belief $p$ in $C(a)$.

The idea of DISPONTE is to associate independent Boolean random variables to the probabilistic axioms. To obtain a world $w$, we include every formula obtained from a certain axiom while we decide whether to include each probabilistic axiom or not in $w$. A world so built is a non probabilistic KB that can be assigned a semantics in the usual way, where a query is entailed if it is true in every model of the world. By multiplying the probability of the choices made to obtain a world we can assign a probability to it. The probability of a query is then the sum of the probabilities of the worlds where the query holds true.

Example 1. Consider the following KB, inspired by the people+pets ontology [19]:

\[
0.5 :: \exists \text{hasAnimal}.\text{Pet} \subseteq \text{NatureLover} \quad 0.6 :: \text{Cat} \subseteq \text{Pet} \\
\text{(kevin, tom)} : \text{hasAnimal} \quad \text{(kevin, fluffy)} : \text{hasAnimal} \quad \text{tom} : \text{Cat} \quad \text{fluffy} : \text{Cat}
\]

The KB indicates that the individuals that own an animal which is a pet are nature lovers with a 50% probability and that kevin has the animals fluffy and tom. Fluffy and tom are cats and cats are pets with probability 60%. We associate a Boolean variable to each axiom as follow $F_1 = \exists \text{hasAnimal}.\text{Pet} \subseteq \text{NatureLover}$, $F_2 = \text{(kevin, fluffy)} : \text{hasAnimal}$, $F_3 = \text{(kevin, tom)} : \text{hasAnimal}$, $F_4 = \text{fluffy} : \text{Cat}$, $F_5 = \text{tom} : \text{Cat}$ and $F_6 = \text{Cat} \subseteq \text{Pet}$.

The KB has four worlds and the query axiom $Q = \text{kevin} : \text{NatureLover}$ is true in one of them, the one corresponding to the selection $\{(F_1, 1), (F_2, 1)\}$. The probability of the query is $P(Q) = 0.5 \cdot 0.6 = 0.3$. 

3
3 Querying probabilistic KBs

Traditionally, a reasoning algorithm decides whether an axiom is entailed or not by a KB by refutation: the axiom $E$ is entailed if $\neg E$ has no model in the KB. Besides deciding whether an axiom is entailed by a KB, we want to find also explanations for the axiom. The problem of finding explanations for a query has been investigated by various authors [29, 14, 12, 13, 10].

The system BUNDLE [26, 27, 20, 21, 24] computes the probability of a query w.r.t. KBs that follow the DISPONTE semantics. It first computes all the explanations for the query by exploiting the Pellet reasoner [31]. A set of explanations $K$ for a query $Q$ is a set of sets of pairs $(E_i, k)$ where $E_i$ is the $i$th probabilistic axiom and $k \in \{0, 1\}$ indicates whether $E_i$ is chosen to be included in a world $(k = 1)$ or not $(k = 0)$. From $K$, we can build a Disjunctive Normal Form (DNF) Boolean formula $f_K$ as $f_K(X) = \bigvee_{\kappa \in K} \bigwedge_{(E_i, 1)} X_i \bigwedge_{(E_i, 0)} \overline{X}_i$. The variables $X = \{X_i | \exists k(E_i, k) \in \kappa, \kappa \in K\}$ are independent Boolean random variables and the probability that $f_K(X)$ takes on value 1 is equal to the probability of $Q$. BUNDLE builds a Binary Decision Diagram (BDD) that represents this DNF formula. A BDD for a function of Boolean variables is a rooted graph that has one level for each Boolean variable. A node $n$ has two children: one corresponding to the 1 value of the variable associated with the level of $n$, indicated with $\text{child}_1(n)$, and one corresponding to the 0 value of the variable, indicated with $\text{child}_0(n)$. When drawing BDDs, the 0-branch - the one going to $\text{child}_0(n)$ - is distinguished from the 1-branch by drawing it with a dashed line. The leaves store either 0 or 1. An example of BDD is shown in Figure 1. A BDD allows to compute the probability of $Q$ with a dynamic programming algorithm in polynomial time in the size of the diagram [6].

The system TRILL [35] implements the BUNDLE’s inference algorithm in Prolog and compute the probability of a query w.r.t. KBs that follow the DISPONTE semantics. The system TRILL$^P$ [36, 34] is based on TRILL but resolves queries by directly computing a pinpointing formula [2, 3], which is a monotone Boolean formula equivalent to the DNF formula built from the set of explanations by BUNDLE and TRILL, and building the corresponding BDD from which the probability of the query is computed.

Example 2. Let us consider the following knowledge base, similar to the ontology people+pets proposed in example 1:

$\exists \text{hasAnimal}.Pet \sqsubseteq \text{NatureLover}$

$(\text{kevin, fluffy}) : \text{hasAnimal}$

$(\text{kevin, tom}) : \text{hasAnimal}$

$(E_1) 0.4 :: \text{fluffy} : \text{Cat}$

$(E_2) 0.3 :: \text{tom} : \text{Cat}$

$(E_3) 0.6 :: \text{Cat} \sqsubseteq \text{Pet}$

Individuals that own an animal which is a pet are nature lovers and kevin owns the animals fluffy and tom. We believe in the fact that fluffy and tom are cats and that cats
are pets with the specified probability. This KB has eight worlds and the query axiom \( Q = \text{kevin} : \text{NatureLover} \) is true in three of them, corresponding to the following choices: \( \{(E_1, 1), (E_2, 0), (E_3, 1)\}, \{(E_1, 0), (E_2, 1), (E_3, 1)\}, \{(E_1, 1), (E_2, 1), (E_3, 1)\} \). The probability is therefore \( P(Q) = 0.4 \cdot 0.7 \cdot 0.6 + 0.6 \cdot 0.3 \cdot 0.6 + 0.4 \cdot 0.3 \cdot 0.6 = 0.348 \). If we associate the random variables \( X_1 \) to the axiom \( E_1 \), \( X_2 \) to \( E_2 \) and \( X_3 \) to \( E_3 \), the DNF formula representing the set of explanations is \( f(X) = (X_1 \land X_3) \lor (X_2 \land X_3) \). The corresponding BDD is shown in Figure 1.

![BDD for Example 2](image)

**Fig. 1.** BDD for Example 2. It correspond to the function \( f(X) = (X_1 \land X_3) \lor (X_2 \land X_3) \).

### 4 Parameter Learning of Probabilistic DLs

EDGE [22, 23] can learn parameters of probabilistic ontologies under the DISP-ONTE semantics. It is inspired by the algorithm EMBLEM [5, 4], which was developed for learning the parameters of probabilistic logic programs under the distribution semantics. EDGE learns the epistemic probabilities previously introduced by using an Expectation Maximization (EM) algorithm [7], that tries to maximize the likelihood.

EDGE takes as input a DL KB \( K \) and a number of examples that represent the queries. Typically, the queries are concept assertions and are divided into positive examples, representing true information for which we would like to get high probability, and negative examples, representing false information for which we would like to get low probability. EDGE first uses BUNDLE for computing, for each query \( Q \), the BDD encoding its explanations.

For negative examples, EDGE first tries to compute the explanations of the negation of the query, for example, if the negative example is \( a : C \), EDGE tries to execute the query \( a : \neg C \). If no explanations are found, EDGE computes the query \( a : C \), finds the BDD and then negates it.

EDGE main procedure consists of the EM cycle in which the steps of Expectation and Maximization are repeated until the log-likelihood (LL) of the examples reaches a local maximum. At each iteration the LL of the example increases, i.e., the probability of positive examples increases and that of negative examples decreases. The EM algorithm is guaranteed to find a local maximum, which however may not be the global maximum. The details of the procedures can be found in [5].
5 Structure Learning of Probabilistic DLs

LEAP [25] performs structure and parameter learning of probabilistic ontologies under the DISPONTE semantics by exploiting CELOE [15] for the structure, and EDGE (Section 4) for the parameters.

CELOE stands for “Class Expression Learning for Ontology Engineering” and is available in the Java open-source framework DL-Learner for OWL and DLs. It takes as input a KB $\mathcal{K}$, a class Target whose formal description we want to learn, and a set of positive and negative examples (i.e. individuals) or a set of positive only examples. CELOE finds a set of $n$ candidates class expressions $C_i$ ($1 \leq i \leq n$) for adding axioms of the form $\text{Target} \equiv C_i$ or $\text{Target} \sqsubseteq C_i$ and sorts them according to a heuristic.

CELOE is a top-down algorithm that starts from the $\top$ concept and uses the $\mathcal{ALCQ}$ refinement operator defined in [16]. Each generated class expression is evaluated using one of five available heuristics, whose resulting value is used to guide the search in the learning process.

LEAP main procedure first generates a set of class expressions by using CELOE, then it creates assertional axioms, which represent the examples (i.e. queries) for EDGE, using the sets of positive and negative examples. Finally, EDGE is applied to the KB to compute the initial value of the parameters and of the LL. At this point, the learning algorithm starts. First, LEAP performs a greedy search in the space of theories by means of the following steps: for each element of the class expressions set, one probabilistic subsumption axiom at a time of the form $p :: \text{Target} \sqsubseteq C_i$ is added to the ontology $\mathcal{K}$ where $p$ is initialized to the accuracy returned by CELOE. After each addition, LEAP executes EDGE on the extended theory to compute the new value of $LL$ and to updated the parameters of the KB. If $LL$ is better than the current best $LL_0$, the new axiom and the updates of the parameters are kept in the knowledge base, otherwise they are discarded. After testing all the class expressions, the final theory so created is returned to the user.

LEAP is a client-server Java RMI application. The server side contains a class called EDGERemote, which performs the EDGE algorithm. The client side, instead, runs a modified version of CELOE called ProbCELOE and a class called EDGE that invokes the remote methods of EDGERemote in order to compute the log-likelihood and the parameters. Figure 2 illustrates the communication between the LEAP client and the server.

6 Related Work

In [18] the authors presented cr$\mathcal{ALC}$, an extension of $\mathcal{ALC}$ that adopts an interpretation-based semantics. cr$\mathcal{ALC}$ allows statistical axioms of the form $P(C|D) = \alpha$, which means that for any element $x$ in $D$, the probability that it is in $C$ given that is in $D$ is $\alpha$, and of the form $P(R) = \beta$, which means that

\[ \text{http://dl-learner.org/Projects/DLLearner} \]
for each couple of elements $x$ and $y$ in $D$, the probability that $x$ is linked to $y$ by the role $R$ is $\beta$. CR\text{ALC} does not allow to express a degree of belief in axioms. A CR\text{ALC} KB $K$ can be represented as a directed acyclic graph $G(K)$ in which a node represents a concept or a role and the edges represent the relations between them. In [18] the authors presented also a system for learning parameters and structure of CR\text{ALC} knowledge bases. The algorithm starts from positive and negative examples for a single concept and learns a probabilistic definition for the concept. For a set of candidate definitions, their parameters are learned using an EM algorithm. Differently from us, the expected counts are computed by resorting to inference in the graph, while we exploit the BDD structures.

The algorithm GoldMiner, presented in [33, 8], uses a different approach that exploits Association Rules (AR) for building ontologies. GoldMiner builds two transaction tables, one for individuals and one for couples of individuals, starting from data about individuals, named classes and roles extracted using SPARQL queries. The first table contains a row for each individual and a column for all named classes and classes of the form $\exists R.C$, where $R$ is a role and $C$ is a named class. The cells of the table contain 1 if the individual belongs to the class of the column. The second table contains a row for each couple of individuals and a column for each named role. The cells contain 1 if the couple of individual belongs to the role in the column. Finally, the APRIORI algorithm [1] is applied to each table in order to find ARs. These are implications of the form $A \Rightarrow B$ where $A$ and $B$ are conjunctions of columns. Each AR can thus be converted to a subclass or subrole axiom $A \sqsubseteq B$. So, from the learned ARs, a knowledge base can be obtained. Moreover, each AR $A \Rightarrow B$ is associated with a confidence that can be interpreted as the probability of the axiom $p :: A \subseteq B$. So GoldMiner can be used to obtain a probabilistic knowledge base.

Fig. 2. LEAP as a client-server Java RMI application.
7 Future work

Our work aims at developing fast algorithms for managing uncertain information defined in KBs following the probabilistic DISPONTE semantics. The tests made on EDGE, presented in [25], show that it achieves better results in a comparable or smaller time than an approach that exploits Association Rules (ARs). Moreover, [25] presented also a preliminary test for LEAP. Further tests we made on larger datasets showed that LEAP needs very large amount of memory. Thus, we are currently studying improvements for the two learning algorithms and for BUNDLE in order reduce the used memory and, consequently, to improve the scalability in the number of considered examples and in the size of the considered KB. In particular, we are working on several optimizations for all the algorithms and on the application of a Map-Reduce approach to EDGE for reducing the memory consumption and parallelizing the process of building the BDDs and computing the expectation.

In the Map-Reduce approach two operators, called map and reduce, are executed one or more times. The map operation applies a transformation on the input data. Generally, this operation is spread on a number of different processors and/or machines, in order to parallelize the computation. The reduce operation aggregates the results returned by the mapping phase.

Since the memory is used mostly for storing the BDDs, the main idea is to divide the examples given to EDGE in chunks and assign to each processor of each machine one or more of them to each processor of each machine. Building a BDD for an example is independent from building it for each other examples. In this way, the number of examples we can handle should increase. Moreover, due to the parallelization, the execution time should decrease.

Then, we also plan to divide the expectation and the maximization phases so that the expectation step will be computed by the map operator, while the maximization step will be computed by the reduce operator. The main problem is that the map processes that built the BDDs should keep them in main memory through the iterations of the EM algorithm. Moreover, the map processes should send and receive the updated parameters and information through the network rather than write and read them from disk.

Furthermore, we plan to study the possibility of the application of this approach also to LEAP.

8 Conclusions

In this paper we presented two algorithms for learning the parameters and for learning both the structure and the parameters of KBs that follow the DISPONTE semantics. EDGE exploits the BDDs that are built during inference to efficiently compute the expectations for hidden variables using an EM algorithm for learning the parameters. LEAP learns the structure by first performing a search in the space of promising axioms, found by exploiting CELOE, and then a greedy search in the space of the ontologies. After that, the probabilities of
the new axioms are computed by EDGE. For the future, we plan to apply optimization to our algorithms in order to achieve more scalability and make them more competitive.

References


