Learning Probabilistic Description Logics

Fabrizio Riguzzi[1](✉), Elena Bellodi[2], Evelina Lamma[2], Riccardo Zese[2], and Giuseppe Cota[2]

[1] Dipartimento di Matematica e Informatica, University of Ferrara, Via Saragat 1, 44122 Ferrara, Italy
fabrizio.riguzzi@unife.it

[2] Dipartimento di Ingegneria, University of Ferrara, Via Saragat 1, 44122 Ferrara, Italy
{elena.bellodi,evelina.lamma,riccardo.zese}@unife.it, giuseppe.cota@student.unife.it

Abstract. We consider the problem of learning both the structure and the parameters of Probabilistic Description Logics under the DISPONTE semantics. DISPONTE is based on the distribution semantics for Probabilistic Logic Programming and assigns a probability to assertional and terminological axioms. The system EDGE, given a DISPONTE knowledge base (KB) and sets of positive and negative examples in the form of concept assertions, returns the value of the probabilities associated with axioms. We present the system LEAP that learns both the structure and the parameters of DISPONTE KBs explotiting EDGE. LEAP is based on the system CELOE for ontology engineering and exploits its search strategy in the space of possible axioms. LEAP uses the axioms returned by CELOE to build a KB so that the likelihood of the examples is maximized. We present experiments showing the potential of EDGE and LEAP.

1 Introduction

Recently, the problem of representing uncertainty in Description Logics (DLs) has received an increasing attention due to the ubiquity of uncertain information in real world domains. Various authors have studied the use of probabilistic DLs and many proposals have been presented for allowing DLs to represent uncertainty [10,13,16,17,28].

In addition, some works have started to appear about learning the probabilities or the whole structure of probabilistic ontologies. These arise, on one hand, from the fact that specifying the values of the probabilities is a difficult task for humans and data is usually available that could be leveraged for tuning them, and, on the other hand, from the fact that in some domains there exist poor-structured knowledge bases which could be improved [13,14]. A knowledge base with a refined structure and instance data coherent with it allows more powerful reasoning, better consistency checking and improved querying possibilities.

In [2,18,19,23] we proposed an approach for the integration of probabilistic information in DLs called DISPONTE for "DIstribution Semantics for

F. Bobillo et al. (Eds.): URSW 2011-2013, LNAI 8816, pp. 63–78, 2014.
DOI: 10.1007/978-3-319-13413-0_4

Probabilistic ONTologiEs". DISPONTE applies the distribution semantics for probabilistic logic programming [25] to DLs.

In this paper we present an approach for learning the structure of probabilistic DLs following the DISPONTE semantics. The approach is based on the algorithm EDGE for "Em over bDds for description loGics paramEter learning" [21,22] that starts from examples of instances and non-instances of concepts and learns the parameters of a probabilistic theory. EDGE builds Binary Decision Diagrams (BDDs) for representing the explanations of the examples from the theory. The parameters are then tuned using an EM algorithm [8] in which the required expectations are computed directly on the BDDs in an efficient way.

The algorithm for learning the structure is called LEAP for "LEArning Probabilistic description logics" and combines the learning system CELOE with EDGE. The former provides a method to build new (equivalence and subsumption) axioms that can be added to the KB, the latter is used to learn the parameters of these probabilistic axioms.

We provide a performance evaluation of both EDGE and LEAP. For EDGE, we extend the evaluation of [21] with a new dataset and that of [22] by including a cross-validation result. For LEAP, we present a comparison between a theory before and after applying LEAP. The experiments with EDGE show that it achieves statistically significant greater areas under the Precision Recall and the Receiver Operating Characteristics curves (AUCPR and AUCROC) with respect to a theory where the probabilities are obtained from an Association Rule learner. The experiments with LEAP show that it improves the AUCPR and AUCROC of the theory with the difference being statistically significant for AUCROC.

The paper is organized as follows. Section 2 introduces Description Logics and the DISPONTE semantics. Section 3 describes EDGE. In Sect. 4 we introduce LEAP. Section 5 discusses related works and Sect. 6 shows the results of experiments for both systems. Section 7 concludes the paper.

2 Description Logics and the DISPONTE Semantics

Description Logics (DLs) are knowledge representation formalisms that are particularly useful for representing ontologies. Their syntax is usually based on concepts and roles. A concept corresponds to a set of individuals of the domain while a role corresponds to a set of couples of individuals of the domain. In the following we consider and describe $\mathcal{ALC}$ [26].

Let $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ be sets of *atomic concepts*, *roles* and *individuals*, respectively. *Concepts* are defined by induction as follows. Each $A \in \mathbf{A}$ is a concept and $\bot$ and $\top$ are concepts. If C, $C1$ and $C2$ are concepts and $R \in \mathbf{R}$, then $(C_1 \sqcap C_2)$, $(C_1 \sqcup C_2)$ and $\neg C$ are concepts, as well as $\exists R.C$ and $\forall R.C$. 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* $a \neq b$. A *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

A knowledge base $\mathcal{K}$ is usually assigned a semantics in terms of set-theoretic interpretations and models of the form $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* that assigns an element in $\Delta^{\mathcal{I}}$ to each $a \in \mathbf{I}$, a subset of $\Delta^{\mathcal{I}}$ to each $C \in \mathbf{A}$ and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each $R \in \mathbf{R}$. The mapping $\cdot^{\mathcal{I}}$ is extended to all concepts (where $R^{\mathcal{I}}(x) = \{y|(x,y) \in R^{\mathcal{I}}\}$) as:

$$\begin{aligned} \top^{\mathcal{I}} &= \Delta^{\mathcal{I}} & \qquad \bot^{\mathcal{I}} &= \emptyset \\ (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & (C_1 \sqcap C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\ (C_1 \sqcup C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} & (\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \subseteq C^{\mathcal{I}}\} \\ (\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \cap C^{\mathcal{I}} \neq \emptyset\} \end{aligned}$$

A query over a knowledge base is an axiom for which we want to test the entailment from the knowledge base. The entailment test may be reduced to checking the unsatisfiability of a concept in the knowledge base, i.e., the emptiness of the concept.

DISPONTE applies the distribution semantics [25] to probabilistic ontologies. In DISPONTE a *probabilistic knowledge base* $\mathcal{K}$ is a set of certain and probabilistic axioms: *certain axioms* take the form of regular DL axioms, *probabilistic axioms* take the form $p :: E$, where p is a real number in $[0, 1]$ and E is a DL axiom. The probability p can be interpreted as an *epistemic* probability, i.e., as the degree of our belief in axiom E. A DISPONTE KB defines a distribution over DL KBs called *worlds*. Each world w is obtained by including every certain axiom. For each probabilistic axiom, we decide whether or not to include it in w. A world therefore is a non probabilistic KB that can be assigned a semantics in the usual way. 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.

The system BUNDLE [18–20,23,24] computes the probability of a query w.r.t. ontologies that follow the DISPONTE semantics by first computing all the explanations for the query and then building a Binary Decision Diagram (BDD) that represents them. A *set of explanations* for a query Q is a set of sets of pairs (E_i, k) where E_i is the ith probabilistic axiom and $k \in \{0, 1\}$ indicates whether E_i is chosen to be included in a world ($k = 1$) or not ($k = 0$). Given the set of explanations K for a query Q, we can define the Disjunctive Normal Form (DNF) Boolean formula f_K as $f_K(\mathbf{X}) = \bigvee_{\kappa \in K} \bigwedge_{(E_i,1)} X_i \bigwedge_{(E_i,0)} \overline{X_i}$. The variables $\mathbf{X} = \{X_i|(E_i, k) \in \kappa, \kappa \in K\}$ are independent Boolean random variables and the probability that $f_K(\mathbf{X})$ takes on value 1 is equal to the probability of Q. 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 $child_1(n)$, and one corresponding to the 0 value of the variable, indicated with $child_0(n)$. When drawing BDDs, the 0-branch - the one going to $child_0(n)$ - is distinguished from the 1-branch by drawing it with a dashed line. The leaves store either 0 or 1.

Explanations are found by using the Pellet reasoner [27] and are then translated into a BDD that allows to compute the probability of Q with a dynamic programming algorithm in polynomial time in the size of the diagram [7].

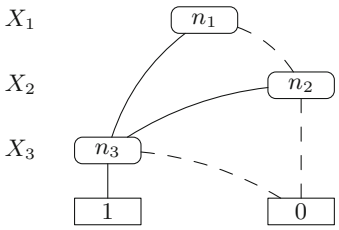

Fig. 1. BDD for Example 1.

The system TRILL [29] implements the BUNDLE's inference algorithm in Prolog and compute the probability of a query w.r.t. KBs that follow the DISPONTE semantics.

Example 1. Let us consider the following knowledge base, inspired by the ontology `people+pets` proposed in [15]:

$$\exists hasAnimal.Pet \sqsubseteq NatureLover$$
$$(kevin, fluffy) : hasAnimal$$
$$(kevin, tom) : hasAnimal$$
$$(E_1)\ 0.4 :: fluffy : Cat$$
$$(E_2)\ 0.3 :: tom : Cat$$
$$(E_3)\ 0.6 :: Cat \sqsubseteq Pet$$

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 $Q = kevin : NatureLover$ 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 BDD representing the set of explanations is shown in Fig. 1.

3 Parameter Learning of Probabilistic DLs

EDGE [21,22] performs parameter learning of probabilistic ontologies under the DISPONTE semantics and is inspired by the algorithm EMBLEM [3,4], which was developed for learning the parameters of probabilistic logic programs under the distribution semantics. The parameters correspond to the epistemic probabilities previously introduced and are tuned using an Expectation Maximization (EM) algorithm [8], an iterative method to estimate some unknown parameters Θ of a model: in particular, it finds maximum likelihood or maximum a posteriori (MAP) estimates of Θ. EM alternates between performing an Expectation (E)

step, where the missing data are estimated given the observed data and current estimate of the model parameters, and a Maximization (M) step, which computes the parameters maximizing the likelihood of the data given the sufficient statistics on the data computed in the E step.

EDGE takes as input a DL KB and a number of examples that represent the queries. Tipically, the queries are concept assertions and are divided into *positive* examples (set E^+) - representing true information, for which we would like to get high probability - and *negative* examples (set E^-) - representing false information, for which we would like to get low probability. EDGE first computes, for each query Q, the BDD encoding its explanations using the reasoner BUNDLE. A limit on the maximum number of explanations to be found ($NumE$) or a time limit for the search for explanations (TLE) can be possibly set for BUNDLE. For negative examples, EDGE computes the explanations of the query, builds the BDD and then negates it. For example, if the negative example is $a : C$, EDGE executes the query $a : C$, finds the BDD and then negates it. Given the knowledge base of Example 1 and the positive example $kevin : NatureLover$, we obtain the BDD in Fig. 1.

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, as shown in Algorithm 1. 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. Procedure Expectation returns the LL of the data that is used in the stopping criterion: EDGE stops when the difference between the LL of the current iteration and the one of the previous iteration (LL_0) drops below a threshold ϵ or when this difference is below a fraction δ of the LL.

Algorithm 1. Function EDGE.

```
1: function EDGE(K, E+, E−, ε, δ, NumE, TLE)
2:     Build BDDs    ▷ BUNDLE builds all the BDDs according to the limits NumE and TLE
3:     LL = −inf
4:     repeat
5:         LL_0 = LL
6:         LL = Expectation(BDDs)
7:         Maximization
8:     until LL − LL_0 < ε ∨ LL − LL_0 < −LL · δ
9:     return (LL, K)
10: end function
```

Procedure Expectation (shown in Algorithm 2) takes as input a list of BDDs, one for each example Q, and computes the expectations $\mathbf{E}[c_{i0}|Q]$ and $\mathbf{E}[c_{i1}|Q]$ for all axioms E_i directly over the BDDs. Let c_{ix} be the number of times a Boolean random variable X_i takes on value x for $x \in \{0, 1\}$:

$$\mathbf{E}[c_{ix}|Q] = P(X_i = x|Q).$$

Then it sums up the contributions of all examples: $\mathbf{E}[c_{ix}] = \sum_Q \mathbf{E}[c_{ix}|Q]$. Finally, $P(X_i = x|Q)$ is given by $\frac{P(X_i=x,Q)}{P(Q)}$. In this procedure we use $\eta^x(i)$

Algorithm 2. Function Expectation.

```
1: function EXPECTATION(BDDs)
2:     LL = 0
3:     for all i ∈ Axioms do
4:         E[c_i0] = 0; E[c_i1] = 0
5:     end for
6:     for all BDD ∈ BDDs do
7:         for all i ∈ Axioms do
8:             η^0(i) = 0; η^1(i) = 0
9:         end for
10:        for all variables X do
11:            ς(X) = 0
12:        end for
13:        GETFORWARD(root(BDD))
14:        Prob=GETBACKWARD(root(BDD))
15:        T = 0
16:        for l = 1 to levels(BDD) do
17:            Let X_i be the variable associated with level l
18:            T = T + ς(X_i)
19:            η^0(i) = η^0(i) + T × (1 − p_i)
20:            η^1(i) = η^1(i) + T × p_i
21:        end for
22:        for all i ∈ Axioms do
23:            E[c_i0] = E[c_i0] + η^0(i)/Prob
24:            E[c_i1] = E[c_i1] + η^1(i)/Prob
25:        end for
26:        LL = LL + log(Prob)
27:    end for
28:    return LL
29: end function
```

to indicate $P(X_i = x, Q)$. Expectation first calls procedures GetForward and GetBackward that compute the forward and the backward probability of nodes and $\eta^x(i)$ for non-deleted paths only. These are the paths that have not been deleted when building the BDDs. Forward and backward probabilities in each node represent the probability mass of paths from the root to the node and that of the paths from the node to the leaves respectively. The expression

$$P(X_i = x, Q) = \sum_{n \in N(Q), v(n)=X_i} F(n)B(child_x(n))\pi_{ix},$$

with $N(Q)$ the set of BDD nodes for query Q, $v(n)$ the variable associated with node n, $\pi_{i1} = p_i$, $\pi_{i0} = 1 - p_i$, $F(n)$ the forward probability of n, $B(n)$ the backward probability of n, represents the probability mass of each path passing through each node associated with X_i and going down its x-branch. We use the notation $e^x(n)$ to indicate the expression inside the sum. Computing the two types of probability in the nodes requires two traversals of the graph, so its cost is linear in the number of nodes.

Procedure GetForward computes the value of the forward probabilities for every node. It traverses the diagram one level at a time starting from the

root level, where $F(root) = 1$, and for each node n computes its contribution to the forward probabilities of its children. Then the forward probabilities of both children are updated. Function GetBackward computes the backward probability of nodes by traversing recursively the tree from the leaves to the root. It returns the backward probability of the root corresponding to the probability of the query $P(Q)$, indicated as $Prob$ at line 14 of Algorithm 2.

When the calls of GetBackward for both children of a node n return, we compute the $e^x(n)$ and $\eta^x(i)$ values for non-deleted paths. An array ς is used to store the contributions of the deleted paths by starting from the root level and accumulating $\varsigma(l)$ for the various levels l. See [22] for more details.

Expectations are updated for all axioms (lines 23–24) and finally the log-likelihood of the current example is added to the overall log likelihood.

Procedure Maximization computes the parameters values for the next EM iteration by relative frequency for all axioms E_i:

$$p_i = \frac{\mathbf{E}[c_{i1}]}{\mathbf{E}[c_{i0}] + \mathbf{E}[c_{i1}]}.$$

4 Structure Learning of Probabilistic DLs

LEAP performs structure and parameter learning of probabilistic ontologies under the DISPONTE semantics by exploiting: (1) CELOE [11] for the structure, and (2) EDGE (Sect. 3) for the parameters. We first introduce CELOE before describing LEAP.

4.1 CELOE

CELOE [11] stands for "Class Expression Learning for Ontology Engineering" and is available in the Java open-source framework DL-Learner[1] for OWL and DLs.

Let us consider a knowledge base $\mathcal{K}$ and a class `Target` whose formal description we want to learn. `Target` has (inferred or asserted) instances in $\mathcal{K}$. CELOE can take as input a target class, a set of positive and negative examples (i.e. *individuals*) or a set of positive only examples.

If `Target` is already described by a class expression C through axioms such as $\texttt{Target} \sqsubseteq C$ or $\texttt{Target} \equiv C$, it is possible to learn a description for `Target` by refining C or by relearning from scratch, as stated in Definition 1.

Definition 1 (Class Learning Problem). *Let an existing named class* `Target` *be in a knowledge base* $\mathcal{K}$. *Let* $R_{\mathcal{K}}(C)$ *be a retrieval reasoner operation that returns the set of all instances of* C. *The class learning problem is to find an expression* C *such that* $R_{\mathcal{K}}(\texttt{Target}) = R_{\mathcal{K}}(C)$.

CELOE finds a set of n class expressions C_i $(1 \leq i \leq n)$ sorted according to a heuristic. Such expressions are candidates for adding axioms of the form $\texttt{Target} \equiv C_i$ or $\texttt{Target} \sqsubseteq C_i$.

[1] http://dl-learner.org/Projects/DLLearner

On the other hand, if a set of positive and negative examples or a set of only positive examples is given, CELOE can be seen as a learning algorithm that solves a problem of learning from examples, as described in Definition 2.

Definition 2 (Learning from Examples Problem).
Given:

- *a concept name* `Target`*;*
- *a knowledge base* $\mathcal{K}$ *not containing* `Target`*;*
- *a space of possible concepts* $\mathcal{C}$*;*
- *a set of positive examples* E^+ *with elements of the form* a : `Target` $(a \in \mathbf{I})$*;*
- *a set of negative examples* E^- *with elements of the form* a : `Target` $(a \in \mathbf{I})$*;*

Find a concept expression $C \in \mathcal{C}$ *such that:*

- `Target` *does not occur in* C *(acyclic definition);*
- $\forall e^+ \in E^+, \mathcal{K} \cup \{\texttt{Target} \equiv C\} \models e^+$*;*
- $\forall e^- \in E^-, \mathcal{K} \cup \{\texttt{Target} \equiv C\} \not\models e^-$.

If $\mathcal{K}' = \mathcal{K} \cup \{\texttt{Target} \equiv C\}$ we say that a concept C *covers* an example $e \in E^+ \cup E^-$ if $\mathcal{K}' \models e$. We distinguish the two cases in which both sets E^+ and E^- of individuals are given or only the set E^+ is given as *Positive and Negative Examples Learning Problem* (LP) and *Positive Examples Learning Problem* respectively.

CELOE is a top-down algorithm that starts from the $\top$ concept and uses the $\mathcal{ALCQ}$ refinement operator defined in [12]. 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. All these heuristics need a set of examples in order to be computed; in the case the algorithm took as input only the target class to be described, we can consider as positive examples the existing instances (inferred or asserted) of the target class and the remaining instances in the KB as negative examples.

4.2 LEAP

In order to learn an ontology, LEAP first finds good candidate axioms (subsumption axioms) by means of CELOE, then it performs a greedy search in the space of theories.

LEAP main procedure is shown in Algorithm 3: it takes as input the knowledge base $\mathcal{K}$ and the type of learning problem LP_{type}; the maximum number of class expressions $NumC$ and the time limit TLC for CELOE; the values of ϵ and δ, the maximum number of explanations $NumE$ and the time limit TLE for the computation of the BDDs for each example for EDGE. Note that CELOE's default is $NumC = 10$ and $TLC = 10\,\text{s}$ and EDGE's default is $NumE = TLE = \infty$.

In the first phase, a set of class expressions is generated by using CELOE (line 2), then the sets of positive (P_I) and negative (N_I) individuals are extracted according to the following rules:

– if a set of positive and negative individuals has been given as input to CELOE (LP_{type} = *Positive and Negative Examples Learning Problem*), then no extraction is necessary;
– if a set of positive only individuals has been given (LP_{type} = *Positive Examples Learning Problem*), then the set of negative examples will be composed of all the individuals of $\mathcal{K}$ except the positive ones;
– if a target class has been given (LP_{type} = *Class Learning Problem*, cf. Definition 1), then we consider the existing instances (inferred or asserted) of the target class as positive individuals and the remaining instances as negative individuals.

After the extraction, the *assertional* axioms, which represent the examples (i.e. queries) for EDGE, are created (see lines 4–9). Then EDGE is applied to the KB to compute the initial value of the parameters and of the LL.

In the second phase, LEAP performs a greedy search in the space of theories, described in lines 11–19. For each element of the class expressions set, one probabilistic subsumption axiom at a time of the form $p :: CE \sqsubseteq \texttt{Target}$ is added to the ontology $\mathcal{K}$; p is either a random probabilistic value or the accuracy returned by CELOE. After each addition, EDGE is run on the extended theory to compute the log-likelihood of the data LL and the updated parameters (line 14). If LL is better than the current best LL_0, the new axiom is kept in the knowledge base, otherwise the new axiom is discarded (lines 15–18). The final theory, obtained from the union of the initial ontology and the probabilistic subsumption axioms learned, 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.

5 Related Work

GoldMiner [10,28] is an algorithm that exploits Association Rules (ARs) for building ontologies. GoldMiner extracts information about individuals, named classes and roles using SPARQL queries. From these data, it builds two *transaction tables*: one that stores the classes to which each individual belongs and one that stores the roles to which each couple of individuals belongs. Finally, the APRIORI algorithm [1] is applied to each table in order to find ARs. Implications of the form $A \Rightarrow B$ can be converted to subclass axioms of the form $A \sqsubseteq B$. Moreover, the confidence p of an AR can be interpreted as the probability of the axiom $p :: A \sqsubseteq B$. So GoldMiner can be used to obtain a probabilistic knowledge base.

The structure learner LEAP is inspired to SLIPCOVER, an algorithm proposed for learning probabilistic logic programs based on distribution semantics [5]. LEAP shares with it the search strategy and the use of the log-likelihood of the data as the score of the learnt theories. Like SLIPCOVER, it divides the

Algorithm 3. Function LEAP.

```
1: function LEAP(K, LP_type, NumC, TLC, ϵ, δ, NumE, TLE)
2:     ClassExpressions = up to NumC or until TLC is reached        ▷ generated by CELOE
3:     (P_I, N_I) = ExtractIndividuals(LP_type)
4:     for all ind ∈ P_I do                                         ▷ P_I: set of positive individuals
5:         Add ind : Target to P_E                                  ▷ P_E: set of positive examples
6:     end for
7:     for all ind ∈ N_I do                                         ▷ N_I: set of negative individuals
8:         Add ind : Target to N_E                                  ▷ N_E: set of negative examples
9:     end for
10:    (LL_0, K) = EDGE(K, P_E, N_E, ϵ, δ, NumE, TLE)
11:    for all CE ∈ ClassExpressions do
12:        Axiom = p::CE ⊑ Target
13:        K' = K ∪ {Axiom}
14:        (LL, K') = EDGE(K', P_E, N_E, ϵ, δ, NumE, TLE)
15:        if LL > LL_0 then
16:            K = K'
17:            LL_0 = LL
18:        end if
19:    end for
20:    return K
21: end function
```

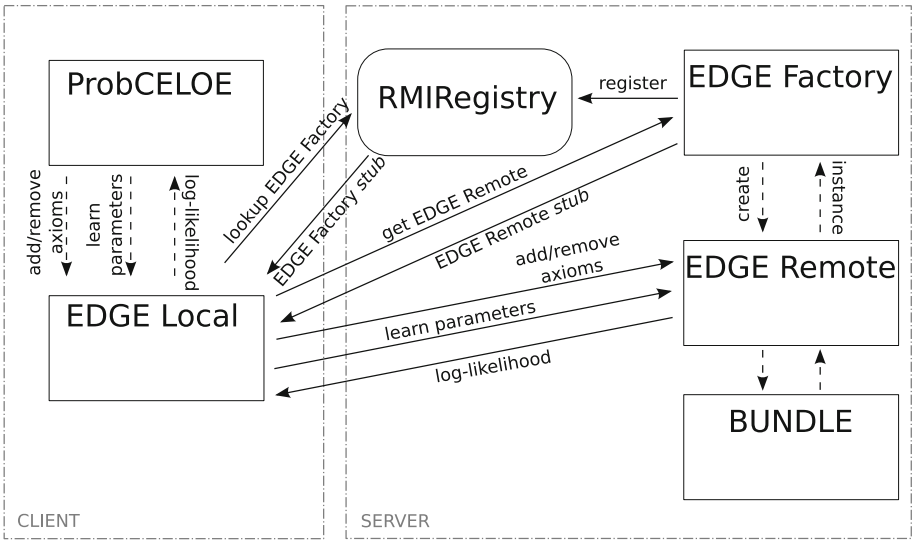

Fig. 2. LEAP as a client-server Java RMI application.

search between learning promising axioms and building in a greedy way a theory whose parameters are optimized by relying on a parameter learning algorithm.

A work that integrates parameter and structure learning for a probabilistic extension of $\mathcal{ALC}$, named CR$\mathcal{ALC}$, is [13]. CR$\mathcal{ALC}$ allows statistical axioms of the form $P(C|D) = \alpha$, meaning that for any element x in $\mathcal{D}$, the probability that it is in C given that is in D is α, and of the form $P(R) = \beta$, meaning that for each couple of elements x and y in $\mathcal{D}$, the probability that x is linked to y by the role R is β. CR$\mathcal{ALC}$ does not allow to express a degree of belief in axioms

as DISPONTE. An algorithm is presented in [13] that learns parameters and structure of CR$\mathcal{ALC}$ KBs. It starts from positive and negative examples for a single concept and from the general concept $\top$ in the root of a search tree to be refined. For a set of candidate concept definitions, their probabilistic parameters are learned using an EM algorithm and a score is assigned to the corresponding node. If the best score in the tree is above a threshold, a deterministic concept definition is returned, otherwise a probabilistic inclusion C_i is searched on a weighted spanning tree, where the target concept is added as a parent of each vertex and probabilities are learned as $P(C_i|Parents(C_i))$. We share the top-down procedure for building axioms (CELOE) but we exploit the BDD structures instead of resorting to inference in a graphical model to compute the expected counts for EM.

The paper [14] presents a Statistical Relational Learning system for learning terminological naïve Bayesian classifiers, which estimate the probability that an individual a belongs to a certain target concept given its membership to a set of induced DL (feature) concepts. The classifier consists of a Bayesian Network (BN) modelling the dependency relations between the feature concepts and the target one. The learning process handles three different assumptions that can be made about the lack of knowledge (under OWA) regarding concept-membership, reflecting in the adoption of different scoring functions and search strategies of the optimal network and parameters. Under one of the assumptions - the probability of concept-membership of a depends on the knowledge on a available in $\mathcal{K}$ - the EM method is proposed to train the BN parameters. The classifier can be seen as a learner of probabilistic assertional axioms, while LEAP learns probabilistic terminological axioms. We exploit BDDs instead of BNs, while we share with them the use of EM.

6 Experiments

In order to test the performances of EDGE and of LEAP we performed several experiments. First we have executed two tests on EDGE which are inspired by the ones presented in [21,22]. Then, once shown that EDGE achieves good results, we have done a preliminary test for comparing LEAP with it.

6.1 Parameter Learning

EDGE has been compared with Association Rules (ARs) [21,22] over two real world datasets from the Linked Open Data cloud: education.data.gov.uk[2] and an extract of DBPedia[3]. We extend the experiments from [21] by including the DBPedia dataset and those of [22] by presenting the results of a cross-validation rather than of a single training-test split.

In the experiments, we wanted to simulate the situation in which an expert provides the structure of the ontology together with information on a set of

[2] http://education.data.gov.uk/
[3] http://dbpedia.org/About

individuals. The ontologies were obtained with GoldMiner: we extracted 10,000 individuals and 5,545 axioms for education.data.gov.uk and 7,200 individuals and 6,228 axioms for DBPedia and we learned ARs from the resulting transaction tables. The ARs were then converted into subclass axioms.

In order to generate a set of examples (queries) for EDGE, for each extracted individual a we sampled three named classes: A and B were sampled from the named classes to which a explicitly belonged, while C was sampled from the named classes to which a did not explicitly belong but that exhibited at least one explanation for the query $a : C$. The axiom $a : A$ is added to the KB, while $a : B$ is considered as a positive example and $a : C$ as a negative example. We used a 5-fold cross validation to test the system. In the training phase, we ran EDGE on the ontology obtained by GoldMiner where we consider all the axioms as probabilistic. We randomly set the initial values of the parameters. EDGE, for handling 5,000 examples, took about 15,000 s in average for DBPedia, about 3 s per example, and about 173,000 s in average for education.data.gov.uk, about 43 s per example. Most of the runtime was spent finding the explanations and building the BDDs, while the execution of the EM iterations took only about 6 s for DBPedia and about 2 s for education.data.gov.uk. In the testing phase, we computed the probability of the queries using BUNDLE. For a negative example of the form $a : C$ we compute the probability p of $a : C$ and we assign probability $1 - p$ to the example.

We compare the parameters learned by EDGE with ARs' confidence. For each AR corresponding to the subclass axiom $A \sqsubseteq B$, we computed the confidence by running two SPARQL queries over the training KBs, one for finding all the individuals that belong to $A \sqcap B$ and one for those that belong to A. The confidence is then given by the ratio of the number of individuals in $A \sqcap B$ over those in A. We created 330 different SPARQL queries for education.data.gov.uk and 2,243 for DBPedia.

In the testing phase, we computed the probability of the examples in the test set using BUNDLE, according to the theory learned by EDGE and to the theory composed of the ARs with the confidence as probability. We drew the Precision-Recall (PR) and the Receiver Operating Characteristics (ROC) curves and computed the Area Under the Curve (AUCPR and AUCROC) following the methods of [6,9]. Table 1 shows the AUCPR, the AUCROC, the execution times averaged over the five folds and the p-value of a paired two-tailed t-test at the 5 % significance level of the difference in AUCROC and AUCPR. The times are referred to the learning time for EDGE and to the SPARQL queries execution time for ARs. Note that the elapsed time for EDGE depends on the number of executed queries and the number of different explanations involved in each query, while the elapsed time for ARs depends on the number of classes in the KB. EDGE achieves greater areas in a time that is of the same or lower order of magnitude with respect to ARs. For both areas and KBs, the differences are statistically significant at the 5 % level.

Table 1. Areas under the ROC and PR curves with standard deviation, execution times and p-value of a paired two-tailed t-test at the 5 % significance level for EDGE and Association Rules.

Datasets		EDGE	ARs	p-value
education.data.gov.uk	PR	0.9702 ± 0.0289	0.8804 ± 0.0165	0.0051
	ROC	0.9796 ± 0.0166	0.9158 ± 0.0171	0.0093
	Time (s)	173,528	10,490	
DBPedia	PR	0.9784 ± 0.0483	0.5916 ± 0.0999	0.0013
	ROC	0.9902 ± 0.0219	0.4346 ± 0.1319	0.0007
	Time (s)	14,883	578,420	

6.2 Structure Learning

LEAP has been evaluated on the Carcinogenesis[4] KB which contains 22,372 individuals and 74,405 axioms.

We randomly selected 180 individuals, 103 of which representing positive examples for the class *Compound* (P_E), i.e. individuals that belong to the class *Compound*, and 77 representing negative examples (N_E), i.e. individuals that do not belong to the class *Compound*. We assigned a random probability to every axiom of the KB and we applied a 5-fold cross validation.

In the training phase, we first ran EDGE on the original KB for learning the parameters associated with the probabilistic axioms, with $NumE = 3$ and $TLE = \infty$ for the call to BUNDLE (cf. Algorithm 1) in order to limit the runtime. Then, we separately ran LEAP on the original KB for learning probabilistic subsumption axioms for the class *Compound* and the associated parameters, with $LP_{type} = Positive\ and\ Negative\ Examples\ Learning\ Problem$. LEAP learned 1 probabilistic subsumption axiom. For CELOE, we set $NumC = 3$ and a timeout TLC for its execution of 120 s: when the timeout expires or 3 class expressions are found, the current set of them is returned to the caller. For EDGE, we set $NumE$ and TLE as before.

In the testing phase, we computed the probability of the examples (queries) in the test set according to the KB learnt by LEAP and the original one, by applying BUNDLE. We drew the PR and ROC curves and computed the AUCPR and AUCROC. Table 2 shows the AUCPR and the AUCROC averaged over the folds together with the standard deviation. Table 3 reports the learning time in seconds, most of which was spent by EDGE for computing the explanations of the examples and building the corresponding BDDs.

The p-value of a paired two-tailed t-test of the difference in AUCPR and AUCROC between the LEAP ontology and the initial one is 0.0603 for AUCPR and 0.0360 for AUCROC, thus showing that LEAP can achieve better areas under both the PR and ROC curves, with the difference in AUCROC being statistically significant at the 5 % significance level.

[4] http://dl-learner.org/wiki/Carcinogenesis

Table 2. Results of the experiments in terms of AUCPR and AUCROC averaged over the folds. Standard deviations are also shown.

Original KB		LEAP	
AUCPR	AUCROC	AUCPR	AUCROC
0.534 ± 0.1082	0.4452 ± 0.0510	0.8006 ± 0.2399	0.798 ± 0.2463

Table 3. Learning time in seconds for LEAP, divided into stages. 'Other' refers to the initialization of the systems and the time spent for sending information between ProbCELOE and EDGE.

	Time (s)
ProbCELOE	139
EDGE	1,765
Other	0.206
Total	1,905

7 Conclusions

We have discussed two algorithms for learning the parameters and the structure of probabilistic DLs following the DISPONTE semantics. EDGE applies an EM algorithm for learning the parameters. It exploits the BDDs that are built during inference to efficiently compute the expectations for hidden variables. The experiments over two real world datasets show that EDGE achieves larger areas both under the PR and the ROC curve with respect to an algorithm based on Association Rules in a comparable or smaller time, thus demonstrating that EDGE is a viable alternative to ARs.

LEAP learns the structure by first performing a search in the space of promising axioms, by exploiting CELOE to learn class expressions of target concepts, and then a greedy search in the space of the ontologies. In this second phase the probabilities of the new axioms are computed by EDGE. The experiments over a real world dataset show that LEAP, by learning the target class expressions, achieves larger areas under both the PR and the ROC curve than a single execution of EDGE. Both EDGE and LEAP are available for download from http://sites.unife.it/ml/.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: International Conference on Very Large Data Bases, pp. 487–499. Morgan Kaufmann (1994)
2. Bellodi, E., Lamma, E., Riguzzi, F., Albani, S.: A distribution semantics for probabilistic ontologies. In: Uncertainty Reasoning for the Semantic Web. CEUR Workshop Proceedings, vol. 778, pp. 75–86. Sun SITE Central Europe (2011)

3. Bellodi, E., Riguzzi, F.: Experimentation of an expectation maximization algorithm for probabilistic logic programs. Intelligenza Artificiale **8**(1), 3–18 (2012)
4. Bellodi, E., Riguzzi, F.: Expectation Maximization over binary decision diagrams for probabilistic logic programs. Intell. Data Anal. **17**(2), 343–363 (2013)
5. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theory and Practice of Logic Programming FirstView Articles (to appear, 2014)
6. Davis, J., Goadrich, M.: The relationship between precision-recall and ROC curves. In: International Conference on Machine Learning, pp. 233–240. ACM (2006)
7. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: International Joint Conference on Artificial Intelligence, vol. 7, pp. 2462–2467 (2007)
8. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc. Ser. B **39**(1), 1–38 (1977)
9. Fawcett, T.: An introduction to ROC analysis. Pattern Recogn. Lett. **27**(8), 861–874 (2006)
10. Fleischhacker, D., Völker, J.: Inductive learning of disjointness axioms. In: Meersman, R., et al. (eds.) OTM 2011, Part II. LNCS, vol. 7045, pp. 680–697. Springer, Heidelberg (2011)
11. Lehmann, J., Auer, S., Bühmann, L., Tramp, S.: Class expression learning for ontology engineering. J. Web Semant. **9**(1), 71–81 (2011)
12. Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Mach. Learn. **78**, 203–250 (2010)
13. Ochoa-Luna, J.E., Revoredo, K., Cozman, F.G.: Learning probabilistic description logics: a framework and algorithms. In: Batyrshin, I., Sidorov, G. (eds.) MICAI 2011, Part I. LNCS, vol. 7094, pp. 28–39. Springer, Heidelberg (2011)
14. Minervini, P., d'Amato, C., Fanizzi, N.: Learning probabilistic description logic concepts: Under different assumptions on missing knowledge. In: ACM Symposium on Applied Computing, pp. 378–383. ACM (2012)
15. Patel-Schneider, P.F., Horrocks, I., Bechhofer, S.: Tutorial on OWL (2003)
16. Riguzzi, F., Bellodi, E., Lamma, E.: Probabilistic Datalog+/- under the distribution semantics. In: International Workshop on Description Logics. CEUR Workshop Proceedings, vol. 846. Sun SITE Central Europe (2012)
17. Riguzzi, F., Bellodi, E., Lamma, E.: Probabilistic ontologies in Datalog+/-. In: Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 857, pp. 221–235. Sun SITE Central Europe (2012)
18. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Epistemic and statistical probabilistic ontologies. In: Uncertainty Reasoning for the Semantic Web. CEUR Workshop Proceedings, vol. 900, pp. 3–14. Sun SITE Central Europe (2012)
19. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: BUNDLE: a reasoner for probabilistic ontologies. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 183–197. Springer, Heidelberg (2013)
20. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Computing instantiated explanations in OWL DL. In: Baldoni, M., Baroglio, C., Boella, G., Micalizio, R. (eds.) AI*IA 2013. LNCS, vol. 8249, pp. 397–408. Springer, Heidelberg (2013)
21. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Parameter learning for probabilistic ontologies. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 265–270. Springer, Heidelberg (2013)

22. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Learning the parameters of probabilistic description logics. In: Inductive Logic Programming Late Breaking papers. CEUR Workshop Proceedings, vol. 1187, pp. 46–51. Sun SITE Central Europe (2014)
23. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Probabilistic description logics under the distribution semantics. Semantic Web - Interoperability, Usability, Applicability (to appear, 2014)
24. Riguzzi, F., Lamma, E., Bellodi, E., Zese, R.: Semantics and inference for probabilistic ontologies. In: Popularize Artificial Intelligence Workshop. CEUR Workshop Proceedings, vol. 860, pp. 41–46. Sun SITE Central Europe (2012)
25. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming, pp. 715–729. MIT Press (1995)
26. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artif. Intell. **48**(1), 1–26 (1991)
27. Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. J. Web Semant. **5**(2), 51–53 (2007)
28. Völker, J., Niepert, M.: Statistical schema induction. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 124–138. Springer, Heidelberg (2011)
29. Zese, R., Bellodi, E., Lamma, E., Riguzzi, F.: A description logics tableau reasoner in Prolog. In: Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 1068, pp. 33–47. Sun SITE Central Europe (2013)