Structure Learning with Distributed Parameter Learning for Probabilistic Ontologies

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

[1] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[giuseppe.cota,riccardo.zese,elena.bellodi,evelina.lamma,
fabrizio.riguzzi]@unife.it

Abstract. We consider the problem of learning both the structure and the parameters of Probabilistic Description Logics under DISPONTE. DISPONTE ("DIstribution Semantics for Probabilistic ONTologiEs") adapts the distribution semantics for Probabilistic Logic Programming to Description Logics. The system LEAP for "LEArning Probabilistic description logics" learns both the structure and the parameters of DISPONTE knowledge bases (KBs) by exploiting the algorithms CELOE and EDGE. The former stands for "Class Expression Learning for Ontology Engineering" and it is used to generate good candidate axioms to add to the KB, while the latter learns the probabilistic parameters and evaluates the KB. EDGE for "Em over bDds for description loGics paramEter learning" is an algorithm for learning the parameters of probabilistic ontologies from data. In order to contain the computational cost, a distributed version of EDGE called EDGE^{MR} was developed. EDGE^{MR} exploits the MapReduce (MR) strategy by means of the Message Passing Interface. In this paper we propose the system LEAP^{MR}. It is a re-engineered version of LEAP which is able to use distributed parallel parameter learning algorithms such as EDGE^{MR}.

Keywords: Probabilistic Description Logics, Structure Learning, Parameter Learning, MapReduce, Message Passing Interface.

1 Introduction

In real world domains the information is often uncertain, hence it is of foremost importance to model uncertainty in representations of the world, including Description Logics (DLs).

In [11, 19, 7, 12] the authors studied the use of probabilistic DLs and various approaches for representing uncertainty in DLs.

Moreover, some works have started to appear about learning the probabilistic parameters or the whole structure of probabilistic ontologies. These are motivated, 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 [11, 10]. A knowledge base with a refined structure and instance data coherent with it permits more powerful reasoning, better consistency checking and improved querying possibilities.

In Probabilistic Logic Programming (PLP) various proposals for representing uncertainty have been presented. One of the most successful approaches is the distribution semantics [17]. In [3, 16, 13] the authors proposed an approach to represent probabilistic axioms in DLs called DISPONTE ("DIstribution Semantics for Probabilistic ONTologiEs"), which adapts the distribution semantics for Probabilistic Logic Programming to DLs.

In the field of Probabilistic Inductive Logic Programming the reasoning task is composed by three main issues: 1) *inference*: we want to compute the probability of a query, 2) *parameter learning*: we know the structure (the logic formulas) of the KB but we want to know the parameters (weights) of the logic formulas and 3) *structure learning*: we want to learn both the structure and the parameters.

LEAP [15] for "LEArning Probabilistic description logics" is an algorithm for learning the structure and the parameters of probabilistic DLs following DISPONTE. It combines the learning system CELOE [9] with EDGE [14]. The former, CELOE ("Class Expression Learning for Ontology Engineering"), provides a method to build new (subsumption) axioms that can be added to the KB, while the latter is used to learn the parameters of these probabilistic axioms.

EDGE stands for "Em over bDds for description loGics paramEter learning" and learns the parameters of a probabilistic theory starting from examples of instances and non-instances of concepts. 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 [6] in which the required expectations are computed directly on the BDDs. This algorithm is rather expensive from a computational point of view. In order to efficiently manage larger datasets in the era of Big Data, it is crucial to develop approaches for reducing the learning time. One solution is to distribute the algorithm using modern computing infrastructure such as clusters and clouds.

In order to reduce EDGE running time, we developed EDGE$^{\mathrm{MR}}$ [5]. It represents a distributed implementation of EDGE and uses a simple MapReduce approach based on the Message Passing Interface (MPI).

In this paper we present an evolution of LEAP called LEAP$^{\mathrm{MR}}$ which adapts the LEAP algorithm to use EDGE$^{\mathrm{MR}}$. In addition, due to a software re-engineering effort, it was possible to remove the RMI module used by LEAP. Compared with LEAP, the quality of the solutions found with LEAP$^{\mathrm{MR}}$ does not change, the difference consists in the running time which is reduced thanks to EDGE$^{\mathrm{MR}}$ and to the removal of the RMI module to a lesser extent. To the best of our knowledge there are no other algorithms that perform distributed structure learning of probabilistic DLs.

Implementing learning algorithms able to elaborate data in a distributed way paves the way to the development of useful tools for Semantic Web and Data Mining in the context of Big Data.

The paper is structured as follows. Section 2 introduces Description Logics and summarizes DISPONTE. Sections 3 and 4 briefly describe the EDGE and EDGE$^{\mathrm{MR}}$ algorithms. Section 5 presents LEAP$^{\mathrm{MR}}$. Finally, Section 7 draws conclusions.

2 Description Logics and DISPONTE

Description Logics (DLs) are a family of logic based knowledge representation formalisms which are of particular interest for representing ontologies and for the Semantic Web. For an extensive introduction to DLs we refer to [1, 2].

While DLs are a fragment of first order logic, they are usually represented using a syntax based on concepts and roles. A concept corresponds to a set of individuals while a role corresponds to a set of couples of individuals of the domain. For the sake of simplicity we consider and describe $\mathcal{ALC}$, but the proposed algorithm can work with $\mathcal{SROIQ}(\mathbf{D})$ DLs.

We use $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ to indicate *atomic concepts*, *atomic roles* and *individuals*, respectively. A *role* is an atomic role $R \in \mathbf{R}$. *Concepts* are defined as follows. Each $A \in \mathbf{A}$, $\bot$ and $\top$ are concepts. If C, C_1 and C_2 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 $\forall R.C$.

Let C and D be concepts, R be a role and a and b be individuals, a *TBox* $\mathcal{T}$ is a finite set of *concept inclusion axioms* $C \sqsubseteq D$, while an *ABox* $\mathcal{A}$ is a finite set of *concept membership axioms* $a : C$ and *role membership axioms* $(a, b) : R$. A *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

A KB is usually assigned a semantics using interpretations 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 individual a, a subset of $\Delta^{\mathcal{I}}$ to each concept C and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role R. The mapping $\cdot^{\mathcal{I}}$ is extended to all concepts as:

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

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

DISPONTE [3] ("DIstribution Semantics for Probabilistic ONTologiEs") applies the distribution semantics to probabilistic ontologies [17]. 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. A

DISPONTE KB defines a distribution over DL KBs called *worlds* assuming that the axioms are independent. Each world w is obtained by including every certain axiom plus a subset of chosen probabilistic axioms.

For each probabilistic axiom $p :: E$, we decide whether or not to include E in w. The probability of this choice is p if the probabilistic axiom is included in the world, $1 - p$ otherwise. A world therefore is a non probabilistic KB that can be handled 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 is true.

3 Parameter Learning for Probabilistic DLs

EDGE [14] is a parameter learning algorithm which adapts the algorithm EMBLEM [4], developed for learning the parameters for probabilistic logic programs, to the case of probabilistic DLs under DISPONTE. Inspired by [8], it performs an Expectation-Maximization cycle over Binary Decision Diagrams (BDDs).

EDGE performs supervised parameter learning. It takes as input a DISPONTE KB and a number of positive and negative examples that represent the queries in the form of concept membership axioms, i.e., in the form $a : C$ for an individual a and a class C. Positive examples represent information that we regard as true and for which we would like to get high probability while negative examples represent information that we regard as false and for which we would like to get low probability.

First, EDGE generates, for each query, the BDD encoding its explanations using BUNDLE [16]. For a positive example of the form $a : C$, EDGE looks for the explanations of $a : C$ and encodes them in a BDD. For a negative example of the form $a : \neg C$, EDGE first looks for the explanations of $a : \neg C$, if one or more are found it encodes them into a BDD, otherwise it looks for the explanations of $a : C$, encodes them in a BDD and negates it with the NOT BDD operator. Then, EDGE starts the EM cycle in which the steps of Expectation and Maximization are iterated until a local maximum of the log-likelihood (LL) of the examples is reached. The LL of the examples is guaranteed to increase at each iteration. EDGE stops when the difference between the LL of the current iteration and that of the previous one drops below a threshold ϵ or when this difference is below a fraction δ of the previous LL. Finally, EDGE returns the reached LL and the new probabilities π_i for the probabilistic axioms. EDGE's main procedure is illustrated in Alg. 1.

Procedure Expectation takes as input a list of BDDs, one for each example Q, and computes the expectations $P(X_i = x|Q)$ for all the random Boolean variables X_i in the BDD and for $x \in \{0, 1\}$. According to DISPONTE, each variable X_i is associated with the probabilistic axioms E_i and has value 1 if the axiom E_i is included in the world, 0 otherwise.

Function Maximization computes the parameters' values for the next EM iteration by relative frequency.

Algorithm 1 Procedure EDGE.

```
function EDGE($\mathcal{K}, P_E, N_E, \epsilon, \delta$)          ▷ $P_E, N_E$: positive and negative examples
    Build $BDDs$                                                   ▷ performed by BUNDLE
    $LL = -\infty$
    repeat
        $LL_0 = LL$
        $LL = $ Expectation($BDDs$)
        Maximization
    until $LL - LL_0 < \epsilon \vee LL - LL_0 < -LL_0 \cdot \delta$
    return $LL$, $p_i$ for all $i$          ▷ $p_i$: learned probability of the $i$-th probabilistic axiom
end function
```

Building BDDs is #P-hard [18]. However, BUNDLE is able to handle domains of significant size. The EM phase, instead, has a linear cost in the number of nodes since the Expectation requires two traversals of the diagrams.

EDGE is written in Java, hence it is highly portable. For further information about EDGE please refer to [14].

4 Distributed Parameter Learning for Probabilistic DLs

In this section we briefly describe a parallel version of EDGE that exploits the MapReduce approach in order to compute the parameters. We called this algorithm EDGE$^{\text{MR}}$ [5].

4.1 Architecture and Scheduling

Like most MapReduce frameworks, EDGE$^{\text{MR}}$'s architecture follows a master-slave model. The communication between the master and the slaves is done by means of the Message Passing Interface (MPI), specifically we use the OpenMPI[3] library which provides a Java interface to the native library.

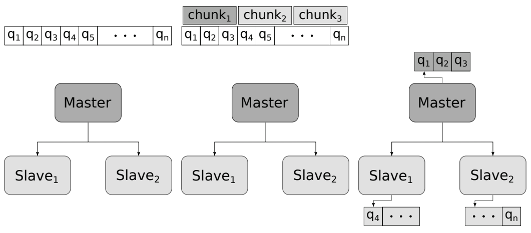
In a distributed context, the performances depend on the scheduling strategy. In order to evaluate different methods, we developed two scheduling strategies: *single-step scheduling* and *dynamic scheduling*. These are used during the queries computation phase.

Single-step Scheduling if N is the number of the slaves, the master divides the total number of queries into $N + 1$ chunks, i.e. the number of slaves plus the master. Then the master begins to compute its queries while, for the other chunks of queries, the master starts a thread for sending each chunk to the corresponding slave. After the master has terminated dealing with its queries, it waits for the results from the slaves. When the slowest slave returns its results to the master, EDGE$^{\text{MR}}$ proceeds to the EM cycle. Figure 1(a) shows an example of single-step scheduling with two slaves.

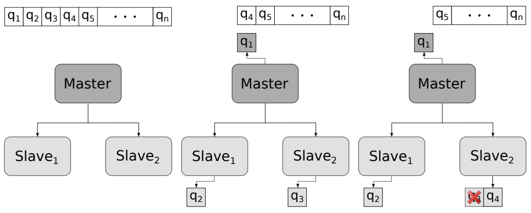
Dynamic Scheduling is more flexible and adaptive than single-step scheduling. Handling each query may require a different amount of time. Therefore, with single-step scheduling, it could happen that a slave takes much more

[3] `http://www.open-mpi.org/`

time than another slave to deal with its chunk of queries. Hence the master and some slaves could be idle. Dynamic scheduling mitigates this issue. The user can establish a chunk dimension, i.e. the number of examples in each chunk. At first, each machine is assigned a chunk of queries in order. Then, if the master ends handling its chunk it just takes the next one, instead, if a slave ends handling its chunk, it asks the master for another one and the master replies by sending a new chunk of queries to the slave. During this phase the master runs a thread listener that waits for the slaves' requests of new chunks and for each request the listener starts a new thread that sends a chunk to the slave which has done the request (to improve the performances this is done through a thread pool). When all the queries are evaluated, EDGE^{MR} starts the EM cycle. An example of dynamic scheduling with two slaves and a chunk dimension of one example is displayed in Fig. 1(b).


chunk$_1$ chunk$_2$ chunk$_3$
q_1 q_2 q_3 q_4 q_5 · · · q_n
Master
Slave$_1$
Slave$_2$
q_4 · · ·
· · · q_n

(a) Single-step scheduling



(b) Dynamic scheduling

Fig. 1. Scheduling techniques of EDGE^{MR}.

Experimental results conducted in [5] show that dynamic scheduling has usually better performances than single-step.

It is obvious that for large sizes of the chunk the dynamic scheduling tends to have the same behavior of single-step. Nevertheless the use of chunks containing only one query can introduce a lot of overhead and therefore reduce the speedup. In order to maximize the speedup it is necessary to find an optimal size of the query chunk.

5 Structure Learning with Distributed Parameter Learning

LEAP^{MR} is an evolution of the LEAP system [15]. While the latter exploits EDGE, the first was adapted to be able to perform EDGE^{MR}. Moreover, after a process of software re-engineering it was possible to remove the RMI communication module used by LEAP and therefore reduce some communication overhead.

It performs structure and parameter learning of probabilistic ontologies under DISPONTE by exploiting: (1) CELOE [9] for the structure, and (2) EDGE^{MR} (Section 4) for the parameters. Figure 2 shows the architecture of LEAP^{MR}.

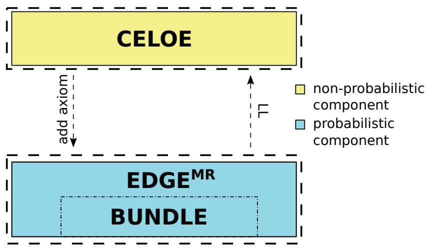


Fig. 2. LEAP^{MR} architecture.

CELOE [9] was implemented in Java and belongs to the open-source framework DL-Learner[4]. Let us consider a knowledge base $\mathcal{K}$ and a concept name `Target` whose formal description, i.e. class description, we want to learn. It learns a set of n class expressions C_i $(1 \leq i \leq n)$ from a set of positive and negative examples. Let $\mathcal{K}' = \mathcal{K} \cup \{C\}$ where $\mathcal{K}$ is the background knowledge, we say that a concept C *covers* an example e if $\mathcal{K}' \models e$. The class expressions found are sorted according to a heuristic. Such expressions can be used to generate candidate axioms of the form $C_i \sqsubseteq$ `Target`.

In order to learn an ontology, LEAP^{MR} first searches for good candidate probabilistic subsumption axioms by means of CELOE, then it performs a greedy search in the space of theories using EDGE^{MR} to evaluate the theories using the log-likelihood as heuristic.

Algorithm 2 shows LEAP^{MR}'s main procedure: it takes as input the knowledge base $\mathcal{K}$ and the configuration settings for CELOE and EDGE^{MR}, then generates $NumC$ class expressions by exploiting CELOE and the sets of positive and negative examples which will be the queries (concept membership axioms) for EDGE^{MR}. A first execution of EDGE^{MR} is applied to $\mathcal{K}$ to compute the initial value of the parameters and of the LL. Then LEAP^{MR} adds to $\mathcal{K}$ one probabilistic subsumption axiom generated from the class expression set at a time. After

[4] `http://dl-learner.org/`

each addition, EDGE$^{\text{MR}}$ is performed on the extended KB to compute the LL of the data and the parameters. If the LL is better than the current best, the new axiom is kept in the knowledge base and the parameter of the probabilistic axiom are updated, otherwise the learned axiom is removed from the ontology and the previous parameters are restored. The final theory is obtained from the union of the initial ontology and the probabilistic axioms learned.

Algorithm 2 Function LEAP$^{\text{MR}}$.

```
1: function LEAP^MR($\mathcal{K}, LP_{type}, NumC, \epsilon, \delta, Schedul$)
2:     $ClassExpressions$ = up to $NumC$                          ▷ generated by CELOE
3:     $(P_I, N_I)$ = ExtractIndividuals($LP_{type}$)   ▷ $LP_{type}$: specifies how to extract $(P_I, N_I)$
4:     for all $ind \in P_I$ 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 $ind \in N_I$ do                                   ▷ $N_I$: set of negative individuals
8:         Add $ind$ : Target to $N_E$                            ▷ $N_E$: set of negative examples
9:     end for
10:    $(LL_0, \mathcal{K})$ = EDGE^MR($\mathcal{K}, P_E, N_E, \epsilon, \delta, Schedul$)   ▷ $Schedul$ : scheduling strategy
11:    for all $CE \in ClassExpressions$ do
12:        $Axiom = p :: CE \sqsubseteq$ Target
13:        $\mathcal{K}' = \mathcal{K} \cup \{Axiom\}$
14:        $(LL, \mathcal{K}')$ = EDGE^MR($\mathcal{K}', P_E, N_E, \epsilon, \delta, Schedul$)
15:        if $LL > LL_0$ then
16:            $\mathcal{K} = \mathcal{K}'$
17:            $LL_0 = LL$
18:        end if
19:    end for
20:    return $\mathcal{K}$
21: end function
```

6 Experiments

In order to test how much the exploitation of EDGE$^{\text{MR}}$ can improve the performances of LEAP$^{\text{MR}}$, we did a preliminary test where we considered the Moral[5] KB which qualitatively simulates moral reasoning. It contains 202 individuals and 4710 axioms (22 axioms are probabilistic).

We performed the experiments on a cluster of 64-bit Linux machines with 8-cores Intel Haswell 2.40 GHz CPUs and 2 GB (max) memory allotted to Java per node. We allotted 1, 3, 5, 9 and 17 nodes, where the execution with 1 node corresponds to the execution of LEAP, while for the other configurations we used the dynamic scheduling with chunks containing 3 queries. For each experiment 2 candidate probabilistic axioms are generated by using CELOE and a maximum of 3 explanations per query was set for EDGE$^{\text{MR}}$. Figure 3 shows the speedup obtained as a function of the number of machines (nodes). The speedup is the ratio of the running time of 1 worker to the one of n workers. We can note that the speedup is significant even if it is sublinear, showing that a certain amount of overhead (the resources, and thereby the time, spent for the MPI communications) is present.

[5] `https://archive.ics.uci.edu/ml/datasets/Moral+Reasoner`

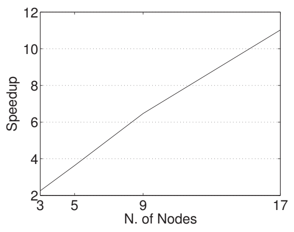


Fig. 3. Speedup of LEAP$^{\mathrm{MR}}$ relative to LEAP for Moral KB.

7 Conclusions

The paper presents the algorithm LEAP$^{\mathrm{MR}}$ for learning the structure of probabilistic description logics under DISPONTE. LEAP$^{\mathrm{MR}}$ performs EDGE$^{\mathrm{MR}}$ which is a MapReduce implementation of EDGE, exploiting modern computing infrastructures for performing distributed parameter learning.

We are currently working for distributing both the structure and the parameter learning of probabilistic knowledge bases by exploiting EDGE$^{\mathrm{MR}}$ also during the building of the class expressions. We would like to distribute the scoring function used to evaluate the obtained refinements. In this function EDGE$^{\mathrm{MR}}$ take as input a KB containing only the individuals and the class expression to test. Finally, the class expressions found are sorted according to the LL returned by EDGE$^{\mathrm{MR}}$ and their initial probability are the probability learned during the execution of EDGE$^{\mathrm{MR}}$.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA (2003)
2. Baader, F., Horrocks, I., Sattler, U.: Description Logics, chap. 3, pp. 135–179. Elsevier Science (2008)
3. Bellodi, E., Lamma, E., Riguzzi, F., Albani, S.: A distribution semantics for probabilistic ontologies. CEUR Workshop Proceedings, vol. 778, pp. 75–86. Sun SITE Central Europe (2011)
4. Bellodi, E., Riguzzi, F.: Expectation Maximization over Binary Decision Diagrams for probabilistic logic programs. Intell. Data Anal. 17(2), 343–363 (2013)
5. Cota, G., Zese, R., Bellodi, E., Lamma, E., Riguzzi, F.: Distributed parameter learning for probabilistic ontologies (2015), to appear
6. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. J. Roy. Stat. Soc. B Met. pp. 1–38 (1977)
7. Fleischhacker, D., Völker, J.: Inductive learning of disjointness axioms. In: On the Move to Meaningful Internet Systems: OTM 2011, pp. 680–697. Springer (2011)

8. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the EM algorithm by BDDs. In: Late Breaking Papers of the International Conference on Inductive Logic Programming. pp. 44–49 (2008)
9. Lehmann, J., Auer, S., Bühmann, L., Tramp, S.: Class expression learning for ontology engineering. J. Web Semant. 9(1), 71–81 (2011)
10. Minervini, P., d'Amato, C., Fanizzi, N.: Learning probabilistic description logic concepts: Under different assumptions on missing knowledge. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing. pp. 378–383. ACM (2012)
11. Ochoa-Luna, J.E., Revoredo, K., Cozman, F.G.: Learning probabilistic description logics: A framework and algorithms. In: Advances in Artificial Intelligence, pp. 28–39. Springer (2011)
12. Riguzzi, F., Bellodi, E., Lamma, E.: Probabilistic Ontologies in Datalog+/-. In: Proceedings of the 9th Italian Convention on Computational Logic, Rome, Italy, June 6-7, 2012. CEUR Workshop Proceedings, vol. 857, pp. 221–235. Sun SITE Central Europe (2012)
13. 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)
14. 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 Berlin Heidelberg (2013)
15. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Learning probabilistic description logics. In: Bobillo, F., Carvalho, R.N., Costa, P.C., d'Amato, C., Fanizzi, N., Laskey, K.B., Laskey, K.J., Lukasiewicz, T., Nickles, M., Pool, M. (eds.) Uncertainty Reasoning for the Semantic Web III, pp. 63–78. LNCS, Springer International Publishing (2014)
16. Riguzzi, F., Lamma, E., Bellodi, E., Zese, R.: BUNDLE: A reasoner for probabilistic ontologies. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 183–197. Springer Berlin Heidelberg (2013)
17. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
18. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comput. 8(3), 410–421 (1979)
19. Völker, J., Niepert, M.: Statistical schema induction. In: The Semantic Web: Research and Applications, pp. 124–138. Springer Berlin Heidelberg (2011)