A PRACTICAL TOOL FOR UNCERTAINTY IN OWL ONTOLOGIES

Shenyong Zhang[1,2], Yi Sun[2], Yun Peng[2], Xiaopu Wang[1]
[1]Department of Astronomy and Applied Physics
University of Science and Technology of China, Hefei, Anhui 230026
[2]Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County, Baltimore, MD 21250
syzhang@mail.ustc.edu.cn, {yisun1, ypeng}@umbc.edu, wxp@ustc.edu.cn

ABSTRACT
Previously we have proposed a theoretical framework, named BayesOWL, which translates an OWL taxonomy of concept classes into a Bayesian network (BN) and incorporates consistent probabilistic information about the concept classes into the translated BN. In this paper, we extend the original framework to support general OWL DL ontologies and to effectively deal with inconsistent probabilistic information. We have also implemented the BayesOWL prototype system, which can be used as a practical tool by people investigating uncertainty in Semantic Web.

KEY WORDS
Semantic Web, Ontology Reasoning, Bayesian Networks

1. Introduction

Typically a Semantic Web ontology is built on concept classes of interest and relationships between these classes of the domain which it intends to model. Semantic Web ontology languages such as OWL and RDF are sufficiently expressive to describe the logical relations such as subclass, subproperty, equivalent, disjoint, etc, between concept classes. However, as logic languages, they cannot quantify and reason with the uncertain aspect of the inter-class relations such as the degree of the overlap or inclusion between two concepts, even if the uncertainty information about the concept classes and inter-class relations is available [1, 2]. For example, one may know 1) concept A is a subclass of B and 2) $P(A|B) = 0.1$. The logical relation of 1) can be represented in OWL, but the probabilistic information in 2) cannot.

In previous work, we have proposed a theoretical framework, named BayesOWL, to address representation and reasoning with uncertainty in Semantic Web ontologies [3, 4]. The framework consists of a set of structure translation rules which are used to translate an OWL taxonomy into a Bayesian network (BN) direct acyclic graph (DAG), and a construction mechanism used to incorporate available probabilistic information about concept classes and interclass relations into the conditional probability tables (CPT) of the BN. The resulting BN is complementary to the original ontology; it models the uncertain aspect of the domain and supports ontological queries by Bayesian reasoning.

Stated more precisely, the BayesOWL takes two inputs:

1) An OWL DL file that defines an ontology *ONT*; and
2) A set of probability distributions **R**, each of which is in the form of $P(u|v)$ where variables in U and V are concept classes in *ONT*.

It outputs a BN, *N*, which

1) Preserves all logical relations among concept classes existing in *ONT*; and
2) Agrees with the probabilistic information in **R** as close as possible.

In the original BayesOWL framework, *ONT* is restricted to simple taxonomical ontologies (i.e. ontologies only consist of concept classes, not anonymous classes). It also requires that probability distributions in **R** be consistent with each other. In this paper, we extend the original framework of BayesOWL to address these limitations. Based on the extended framework we have completed a prototype implementation [5] which is available as a practical tool for researchers and practitioners who are modeling uncertainty in Semantic Web.

The rest of this paper is organized as follows. Section 2 briefly reviews the original BayesOWL structure translation rules and extends it to support general OWL DL ontologies. Section 3 introduces general techniques that integrate inconsistent probabilistic information and shows how these approaches can be applied to BN. Section 4 presents our approach to incorporate probabilistic information, both consistent and inconsistent, to modify the CPTs in the translated BN. Section 5 shows our prototype implementation. And finally Section 6 concludes with discussions and directions for future work.

2. Structure Translation

In this section, we first give a brief review of the structure translation rules in the original BayesOWL, and then discuss how to extend these rules from terminological taxonomies to OWL DL ontologies.

2.1 Original Structure Translation Rules

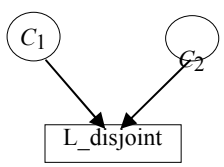
To translate the given *ONT* into a BN, BayesOWL first constructs a DAG, the BN structure, based on the concept class hierarchy. Each concept class *C* in *ONT* is translated into a node *C* in the DAG, treated as a binary random variable with *P*(*C*) as the probability of a randomly chosen individual belonging to class *C*. The arcs between the nodes are set according to the logical relations between these concept classes in *ONT*. Besides the subclass relation, five other logical relations can be specified using OWL constructors [6], they are: equivalent (same as), union, intersection, disjoint, and complement.

The original BayesOWL only deals with OWL ontologies which are simple taxonomies of concepts (i.e., ontologies without OWL properties). Since every class in such taxonomy is a concept class, it is sufficient for the BN DAG to model the subclasses and other logical relations that are explicitly defined in the given ontology file [4]. This is done by

1) Setting an arc from the superclass node to the subclass node for each defined subclass relation and constructing the default CPT of the subclass node;
2) Creating a logic node (L-node for short) for each of the defined logical relation of other five types. Each L-node is connected with the related concept nodes, and its CPT is set according to the designated logical relation.

For a concept node, an entry in its CPT must be zero if any of its parents is 'False' for this entry. The only other entry in the table is the one in which all its parents are 'True'. We set the value of such an entry (0.5, 0.5) as default. As will be seen later, when probabilistic information **R** is provided, these non-zero entries in the CPT for concept nodes will be modified so that the BN will have its distribution satisfying **R**.

For L-nodes, CPT entries are set based on the logical relations they denote. Fig. 1 below gives an example of the L-node for a disjoint relation between concepts C_1 and C_2 and its CPT. It can be seen that, when L_disjoint is set to "True", C_1 and C_2 become disjoint (probability of both being true becomes 0).



C_1	C_2	disjoint	
		T	F
T	T	0.0	1.0
T	F	1.0	0.0
F	T	1.0	0.0
F	F	1.0	0.0

Figure 1. Logic node for "disjoint" and its CPT

An example illustrating the structure translation is given in Fig. 2 below: Fig. 2.(a) gives the ontology of 6 concept classes and logical relations between these concepts; Fig. 2.(b) is the translated DAG where the 6 concept nodes correspond to the 6 concept classes and they form a subclass hierarchy. The 4 defined logical relations are modeled by the 4 L-nodes with their CPTs set according to the respective logical relations.

"Nature" Ontology

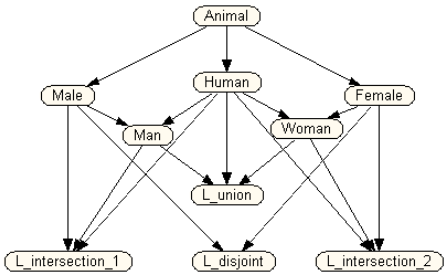
Concept classes:

Animal, Male, Female, Human, Man, Woman;

Defined logical relations:

- **Male, Female**: subclasses of Animal;
- **Male and Female**: disjoint with each other;
- **Man**: intersection of Human and Male;
- **Woman**: intersection of Human and Female;
- **Human**: union of Man and Woman.

(a) A tiny ontology named "Nature"



(b) The translated BN DAG

Figure 2. Taxonomy "Nature" and its BN structure

2.2 Implicit Logical Relations

Modeling logical relations between concept classes in general ontologies is more complicated than those in simple taxonomies. This is because general ontologies may contain anonymous classes and implicit logical relations may exist between concept classes via these anonymous classes. For example, in the Wine ontology provided by W3C as an OWL use example and test benchmark [7], there is no explicitly defined logical relation between two concept classes "RedBurgundy" and "WhiteWine". However, they are disjoint with each other because they are subclasses of two disjoint *anonymous* classes "hasColor = #Red" and "hasColor = #White", respectively.

If we simply exclude these anonymous classes from the translated BN, these implicit logical relations between concept classes will be lost. By using OWL reasoner tools such as Pellet [8], all of these implicit relations can be derived from the given ontology. However, not every of these relations needs to be modeled by an L-node in the translated DAG; some are implied by others and thus are considered redundant.

Let S_{ONT} denote all explicit and implicit logical relations between concept classes of the given ontology, we define $S_{DAG} \subseteq S_{ONT}$ as a set of logical relations such that:

1) every L in S_{ONT} is entailed by S_{DAG}; and
2) no L in S_{DAG} is entailed by $S_{DAG} \setminus \{L\}$.

Then, we only need to model those L in S_{DAG} by subclass arcs and L-nodes in the translated BN. Note that $S_{DAG} = S_{ONT}$ if *ONT* does not contain any anonymous class.

2.3 Removing Redundant Relations

Most of the redundant relations in S_{ONT} are stemmed from the "equivalency", "transitivity", and "commutativity" of logical relations. For example,

- if *equivalent*(A, B) is in S_{ONT}, then for any concept C related to A in S_{ONT}, C is also related to B of the same relation in S_{ONT};
- if $A \subseteq B$ and $B \subseteq C$ are in S_{ONT} then $A \subseteq C$ is also in S_{ONT};
- if $A \subseteq B$ and $B \cap C = \varnothing$ are in S_{ONT} then $A \cap C = \varnothing$ is also in S_{ONT};
- if $A = B \cap C$ is in S_{ONT} then $A = C \cap B$ is also in S_{ONT};

This leads to the following procedure for removing redundant relations.

---

1. **Equivalence**:
   - 1.1 partition the set of all concept classes into equivalence groups, and designate one concept in each group as its representative;
   - 1.2 replace all equivalence relations in S_{ONT} by these equivalence groups: each equivalence group is modeled by a single L_equivalent node and having all concept nodes in that group pointing to this L-node;
   - 1.3 for all remaining relations in S_{ONT}, replace each concept by the representative of the equivalence group it belongs to;
   - 1.4 combine relations that are the same or become the same under commutation into one relation;
2. **Disjoint**: remove $A \cap B = \varnothing$ from S_{ONT} if $C \cap D = \varnothing$, $A \subseteq C$ and $B \subseteq D$ are also in S_{ONT};
3. **Subclass**: for each concept class A
   - 3.1 identify all most specific subsumers of A;
   - 3.2 remove all $A \subseteq C$ from S_{ONT} if C is not a most specific subsumer of A;

Most redundant relations in S_{ONT} will be removed by this procedure. We use the Wine ontology [7] (which also imports Food ontology) to verify our redundancy relation removal procedure. Table 1 shows the results.

Table 1. **Number of relations in "Wine" ontology**

	Explicitly Defined	Derived by Pellet	After Redundant removal
No. of Relations	99	6641	**388**

As given in Table 1, the number of logical relations explicitly defined between concept classes in the ontology file is 99, which is much smaller than the total number of relations derived by Pellet reasoner, 6641. After applying our redundant relation removal procedure, the number is reduced to 388, which is less than 6% of all derivable logical relations. 172 of them are subclass relations, they will be modeled by the subclass arcs in the BN. Each of the rest **216** logical relations is represented by an L-node in the translated BN. The total number of the concept classes in wine ontology is **126**, and the total number of the nodes (the concept nodes plus the L-nodes) in resulting BN is **342**.

2.4 Extended Structure Translation Rules

The rules for structure translation of the extended BayesOWL are now given as follows:

- **Concept Classes**. Each defined concept class is mapped into a binary variable node, called concept node, in the translated BN.
- **Logical Relations**. Construct S_{DAG} by first forming S_{ONT} from the given ontology by OWL reasoner and then applying the redundant relation removal procedure.
- **Subclasses**. For every subclass relation in S_{DAG}, an arc is set from the superclass to the subclass and a default CPT is constructed for the subclass node. A subclass hierarchy of all concept nodes is thus formed.
- **Logical Nodes**. For every logical relation other than subclass in S_{DAG}, create an L-node, connecting all concept nodes in that relation to the L-node, and set its CPT according to the logic of the relation.

When all L-nodes are set to true (denoted *LT*), all logical relations between concept classes in *ONT* hold between the corresponding concept nodes in the translated BN.

Now the task left is to integrate the probabilistic information in **R** (the second input to BayesOWL) into the BN. This is done by modifying the CPTs of the concept nodes so that the joint distribution of the BN, given *LT*, agrees with **R**, i.e.

$$P(u \mid v, LT) = R(u \mid v), \ \forall R(u \mid v) \in \mathbf{R}.$$

This task is dealt with in the next two sections.

3. Probabilistic Information Incorporation

Consider a joint probability distribution (JPD) $Q(X)$ defined on a set of variables $X = \{X_1, ..., X_n\}$ and a set of low-dimensional probability distributions, called constraint $\mathbf{R} = \{R(Y^1), ..., R(Y^m)\}$, where $Y^j \subseteq X, 1 < j < m$.

To incorporate information in **R** into Q is to change $Q(X)$ minimally to $Q^*(X)$ to satisfy all constraints in **R**, i.e.

$$Q^*(y^j) = R_j(y^j),\ \forall R_j(y^j) \in \mathbf{R}.$$

The approach BayesOWL takes for the information integration is based on the method known as *iterative proportional fitting procedure* (IPFP) which iteratively modifies the JPD using the constraints one at a time. If these constraints are consistent with each other, i.e., there exists a JPD that satisfies all constraints, then IPFP will converge. In this section we first briefly describe IPFP, and then discuss how to apply IPFP to modify BN CPTs with probabilistic constraints. Finally we propose our method to deal with inconsistent constraints.

3.1 Brief Introduction to IPFP

Iterative proportional fitting procedure (IPFP) was first published by Kruithof [9]. Deming and Stephan used this procedure on estimating cell frequencies in contingency tables under marginal constraints [10].

IPFP uses I-divergence, also known as Kullback-Leibler divergence [11], relative information or cross-entropy, to measure the distance between two JPDs $P(x)$ and $Q(x)$:

$$I(P \| Q) = \begin{cases} \sum P(x) \log \dfrac{P(x)}{Q(x)} & \text{if } P << Q; \\ +\infty & \text{otherwise} \end{cases}$$

where $P << Q$ (P is dominated by Q) if $\{x \mid P(x) > 0\} \subseteq \{x \mid Q(x) > 0\}$.

Let **P** be a convex set of JPD over X and $Q_0(x) \notin \mathbf{P}$. $Q'(x) \in \mathbf{P}$ is an *I-projection* of $Q_0(x)$ on **P** if it has the minimum I-divergence to $Q_0(x)$ among all JPD in **P**.

For $Q_0(x)$ and $\mathbf{R} = \{R(y^1), ..., R(y^m)\}$ the iterative process of IPFP is defined as following:

$$Q_k(x) = Q_{k-1}(x) \cdot \frac{R(y^i)}{Q_{k-1}(y^i)},$$

where $i = ((k-1) \bmod m) + 1$. IPFP was proved to converge when constraints are consistent, and the converging JPD Q^* is an I-projection of $Q_0(x)$ on the set of all JPDs that satisfy all constraints in **R** [12].

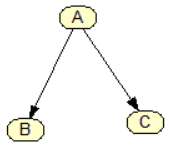
3.2 Modifying BN with Probability Constraints

Applying IPFP to modify the CPT for concept nodes is more complicated. Note that a BN defines a JPD which can be generated by the chain rule

$$Q(x) = \Pi_{i=1}^{n} Q(x_i \mid \pi_i)$$

where X_i is the variable of BN and π_i is the set of all parents of X_i, and the parent-child relations are determined by BN DAG. One would think our task can be accomplished by simply applying IPFP on this JPD and then extracting the CPTs from the resulting JPD Q^* by marginalization. However, this approach would not work because IPFP does not respect the interdependencies between variables dictated by the BN DAG. This can be seen from a simple example in Fig. 3.

On the left of Fig. 3 is the DAG of a BN of three variables *A, B*, and *C*, and its initial CPTs. The JPD $Q(A,B,C)$ (on the upper right) is obtained by modifying the original JPD of this BN with constraint $P(B, C) = (0.4, 0.3, 0.17, 0.13)$. However, if we extract from Q the CPTs $Q(A)$, $Q(B|A)$ and $Q(C|A)$ for the three variables, the new JPD defined by these CPTs, $Q'(A,B,C) = Q(A) \cdot Q(B \mid A) \cdot Q(C \mid A)$ on the lower right, is different from Q, and it does not satisfy constraint $P(B, C)$ any more, although the difference is not that great.



A	B	C	Q(A, B, C)
T	T	T	0.1333
T	T	F	0.1000
T	F	T	0.1253
T	F	F	0.0535
F	T	T	0.2667
F	T	F	0.2000
F	F	T	0.0447
F	F	F	0.0765

A	T	F
	0.5	0.5

A	B	
	T	F
T	0.3	0.7
F	0.6	0.4

A	C	
	T	F
T	0.8	0.2
F	0.5	0.5

A	B	C	Q*(A, B, C)
T	T	T	0.1464
T	T	F	0.0869
T	F	T	0.1122
T	F	F	0.0666
F	T	T	0.2472
F	T	F	0.2195
F	F	T	0.0642
F	F	F	0.0570

Figure 3. A three variable BN example

To resolve this problem, we have developed an algorithm, called E-IPFP, [13], which is an extension of IPFP. E-IPFP forces the process to obey the BN structure by introducing an additional constraint

$$R_{m+1}(x) = \Pi_{i=1}^{n} Q_k(x_i \mid \pi_i)$$

where $Q_k(x_i \mid \pi_i)$ are CPTs extracted from the current JPD during IPFP process.

Let $Q_0(x) = \Pi_{i=1}^{n} Q_0(x_i \mid \pi_i)$ be the initial JPD of the BN and $\mathbf{R} = \{R(Y^1), ..., R(Y^m)\}$ be the set of constraints, E-IPFP is given as following.

---

1. $Q_0(x) = \Pi_{i=1}^{n} Q_0(x_i \mid \pi_i)$;
2. for k = 1 until convergence {
   2.1 $i = ((k - 1) \bmod (m + 1)) + 1$;
   2.2 if $i < m + 1$

$$Q_k(x) = Q_{k-1}(x) \cdot \frac{R(y^i)}{Q_{k-1}(y^i)};$$

   2.3 else

extract $Q_{k-1}(x_i|\pi_i)$ from $Q_{k-1}(x)$ according to BN structure;

$Q_k(x) = \Pi_{i=1}^{n} Q_{k-1}(x_i|\pi_i)$;

2.4 $k = k + 1$;

}

Note that the m constraints in **R** are used in Step 2.2 ($i < m + 1$) in the same ways as the standard IPFP, and the structure constraint $R_{m+1}(x)$ is used in Step 2.3. As proved in [13], E-IPFP converges for consistent probability constraints and the converging JPD satisfies all constraints in **R** and obeys the BN structure.

Like IPFP, E-IPFP is extremely expensive when the BN is large. To reduce the computational cost, we have developed two additional algorithms. The first algorithm, D-IPFP, decomposes the BN into subnets and applies E-IPFP to each subnet with constraints involved in that subnet. The second algorithm, SD-IPFP, is applicable when variables in each constraint are within one CPT. Then each CPT is modified directly by the constraints without generating the JPD. Please refer to [13] for more details of these algorithms.

3.3 Inconsistent Information Integration

As discussed in 3.1, IPFP converges for consistent constraints. But when constraints are inconsistent with each other, IPFP tends to go into cycles [14]. To integrate such inconsistent constraints, the best one can do is to find a JPD whose marginals are as close as possible to the constraints in **R**. Vomlel has proposed an algorithm called GEMA [14, 15], which is a parallel variation of the iterative IPFP. However, the algorithm suffers a high computation cost and its result is very sensitive to the initial JPD and constraint orders [16].

We have developed an efficient method named SMOOTH for inconsistent constraint integration [16]. It is very much like the standard IPFP, except that at each iteration a bi-directional modification takes place: at the k[th] iteration, not only the current JPD $Q_{k-1}(x)$ is modified by the chosen constraint $R(y^j)$, $R(y^j)$ itself is also modified by $Q_{k-1}(y^j)$. Precisely, at the k^{th} iteration we have:

$$R(y^j) = (1-\alpha)Q_{k-1}(y^j) + \alpha R(y^j);$$

$$Q_k(x) = Q_{k-1}(x) \cdot \frac{R(y^j)}{Q_{k-1}(y^j)};$$

where $0 < \alpha < 1$ is the smooth factor, and in order to guarantee these constraints are treated equally, α should be chosen to be very close to 1. It converges with constant α; the convergence can be accelerated by reducing α at the end [16]. Since $Q_{k-1}(x)$ has been modified by other constraints in the previous iterations, using it to modify $R(y^j)$ has the effect of reducing or smoothening the inconsistency among the constraints. Experiments show SMOOTH is data insensitive, time and computation efficient.

Moreover, since it can be seen as an extension of standard IPFP, we can directly apply it to any IPFP-based methods, including E-IPFP.

4. CPT Construction

As discussed in Section 3, a BN translated from OWL ontology contains two kinds of nodes: concept nodes and L-nodes. For L-nodes, CPTs are completely determined by logical operation results, thus the only thing that is left is to construct CPTs for concept nodes. These CPTs shall incorporate the probabilistic information in **R**. When using E-IPFP to construct CPTs for concept nodes, we need to preserve the logical relations among the concept nodes. As shown in Section 2, this is done by setting all L-nodes to true, denoted by *LT*. This requires replacing the formula in Step 2.2 of E-IPFP by

$$Q_k(x) = Q_{k-1}(x) \cdot \frac{R(y^j)}{Q_{k-1}(y^j \mid LT)} \cdot \beta_k,$$

where $\beta_k = 1/\Sigma_x Q_k(x)$ is the normalization factor. If the constraints are inconsistent, then SMOOTH is applied, and

$$R(y^j) = (1-\alpha)Q_{k-1}(y^j \mid LT) + \alpha R(y^j);$$

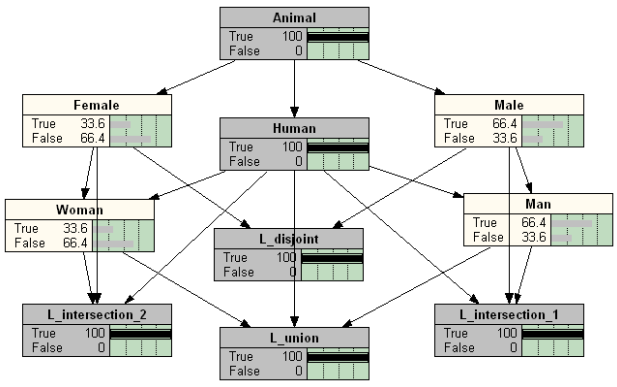
$$Q_k(x) = Q_{k-1}(x) \cdot \frac{R(y^j)}{Q_{k-1}(y^j \mid LT)} \beta_k;$$

shall be used.

As an example, consider the following constraints to the Nature ontology in Fig 2:

- P(Animal) = 0.560
- P(Male,Human|Animal)= 0.511
- P(Female,Human|Animal)= 0.258
- P(Man|Animal,Human)= 0.664
- P(Woman|Animal,Human)= 0.336

These constraints are consistent with each other and with the BN structure. After running E-IPFP to convergence, the translated BN and its final CPTs are given in Fig. 4 below. It can be seen that, after all L-nodes are set to be true, the network is consistent with all the probability constraints.



(a) Resulting BN of the ‘‘Nature” ontology

Animal	
TRUE	FALSE
0.9114	0.0886

Animal	Human	
	TRUE	FALSE
TRUE	0.8341	0.1659
FALSE	0	1

Animal	Female	
	TRUE	FALSE
TRUE	0.8938	0.1062
FALSE	0	1

Animal	Male	
	TRUE	FALSE
TRUE	0.9262	0.0738
FALSE	0	1

Male	Human	Man	
		TRUE	FALSE
TRUE	TRUE	0.7719	0.2281
TRUE	FALSE	0	1
FALSE	TRUE	0	1
FALSE	FALSE	0	1

Female	Human	Woman	
		TRUE	FALSE
TRUE	TRUE	0.5816	0.4184
TRUE	FALSE	0	1
FALSE	TRUE	0	1
FALSE	FALSE	0	1

(b) Final CPTs of the resulting BN

Figure 4. Translation result of the "Nature" ontology

When we change the fourth constraint to

P(Man|Animal,Human)= 0.56,

these constraints become inconsistent with the BN structure in which "Man" and "Woman" are complement with each other. To deal with this inconsistency, SMOOTH version of E-IPFP is applied. At convergence, the translated BN has

P(Man|Animal,Human)= 0.636,

P(Woman|Animal,Human)= 0.364

which are now complement with each other and close to the given constraints. Figure 5 gives the final CPTs for the resulting BN.

Animal	
TRUE	FALSE
0.8007	0.1993

Animal	Human	
	TRUE	FALSE
TRUE	0.8471	0.1529
FALSE	0	1

Animal	Female	
	TRUE	FALSE
TRUE	0.6847	0.3153
FALSE	0	1

Animal	Male	
	TRUE	FALSE
TRUE	0.7628	0.2372
FALSE	0	1

Male	Human	Man	
		TRUE	FALSE
TRUE	TRUE	0.7486	0.2514
TRUE	FALSE	0	1
FALSE	TRUE	0	1
FALSE	FALSE	0	1

Female	Human	Woman	
		TRUE	FALSE
TRUE	TRUE	0.6339	0.3661
TRUE	FALSE	0	1
FALSE	TRUE	0	1
FALSE	FALSE	0	1

Figure 5. Final CPTs of resulting BN (inconsistent)

5. Prototype Implementation

Our prototype implementation [5] of BayesOWL is written in Java. It takes two inputs: 1) an OWL file that defines the ontology which is to be translated into a BN; and 2) an OWL file of the probabilistic information of concepts and inter-concept relations encoded in the way as described in [5]. The translated BN is written in the format of Netica, a BN development software system from Norsys Software Corporation [17]. Netica also provides probabilistic reasoning when constructing the CPTs for the translated BN.

5.1 System Architecture

The architecture of the BayesOWL prototype is given in Figure 6 below. The *Terminological Parser* (*T-Parser*) parses the OWL ontology file, identifies all concept classes defined in the ontology, and generates all logical relations among the concepts.

As discussed in Subsection 2.2, some of these relations are not explicitly defined in the ontology file but can be derived using anonymous classes. To ensure all implicit relations are derived, we choose Pellet [8, 18], a complete reasoner for OWL DL, to help implement the *T-Parser* and construct S_{ONT}. After the S_{ONT} is formed, we remove the redundant relations and generate S_{DAG}.

BN-Structure Constructor takes S_{DAG} as input and outputs the BN DAG. It also constructs the CPTs for all L-nodes and initializes the default CPTs for concept nodes.

The *Probabilistic Parser* (*P-Parser*) parses probability files and extracts the encoded probabilities into a specific internal representation. These probabilities are then taken by *CPT Constructor* as input constraints for E-IPFP algorithm to modify the CPTs of the concept nodes.

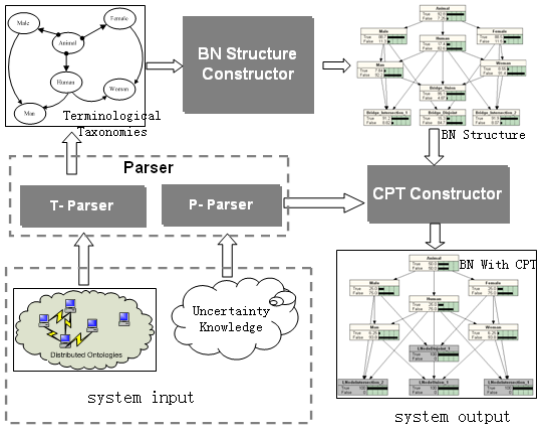


Figure 6. Architecture of BayesOWL prototype system

5.2 The BayesOWL Toolkit

As a practical tool for the users and developers, the BayesOWL prototype provides a toolkit, containing a set of Java APIs and a graphic user interface (GUI). User's reference manual and other related documents are also provided.

The APIs (see Figure 7 below) includes several packages which can be used to complete the ontology-to-BN translation. The packages and the methods included there can also be used separately.

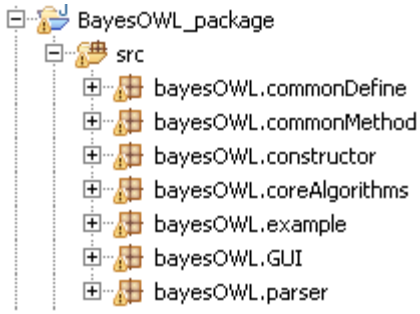


Figure 7. BayesOWL APIs

The system GUI is shown in Figure 8. The layout is divided into several areas:

- **File input area**: which is used to input OWL ontology files and probability files;
- **Options area**: which is designed for optional operations such as requesting Netica license for large BN, the location the resulting BN to be saved, and whether you want to open and view the resulting BN when it is generated;
- **Log area**: which is for showing the running status;
- **Result BN area**: which shows the translated BN in a tree structure; and
- **Node detail area**: which gives node details (including its prior beliefs and its parents) when a node is selected in resulting BN area.

The BayesOWL GUI is executable. After the input ontology and probability files are specified, the "start" button starts the translation, the resulting BN will be generated and saved, and the network structure is shown in the translation result area.

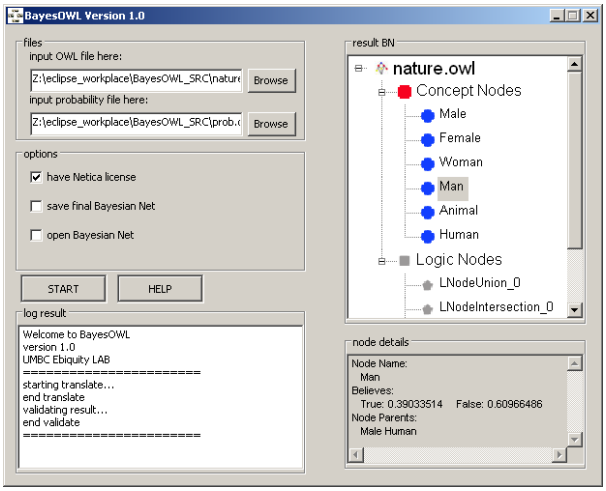


Figure 8. GUI of BayesOWL

6. Conclusions and Future Work

In this paper, we have extended our original BayesOWL framework so that it is able to model the implicit logical relations entailed via anonymous classes, thus to support general OWL DL ontologies, and to incorporate both consistent and inconsistent probabilistic information. These extensions allow one to translate terminological taxonomy of an OWL DL ontology into a BN. As a probabilistic model, the translated BN complements the logic-based Semantic Web languages and supports ontological queries as Bayesian reasoning.

The BayesOWL prototype system and its Java APIs and GUI can be used as a practical tool for those who are interested in representing and reasoning with uncertainty in Semantic Web.

Currently we are intensely working on the theoretical analysis of BayesOWL framework. Recently we have completed more rigorous proofs of convergence of both SMOOTH and E-IPFP. Analysis of the framework's time complexity is underway, with the emphasis on the complexity of the CPT construction since it dominates the computation of the framework.

We are also actively working on identifying real world application scenarios that are best suitable for BayesOWL. Experiments with these scenarios will allow us to evaluate the validity and effectiveness of the framework.

One direction for future work is the automatic generation of probabilistic information. Relying on ontology designers and domain experts may not always be possible and often error prone, especially when the ontology is large. One potential way is to mine the web data using text classification methods [19, 20, 21] and other information retrieval techniques.

Another direction is to extend the BayesOWL framework to properties and individuals, so that a complete uncertainty model for complex OWL DL ontologies can be built.

Acknowledgements

This work was supported in part by NIST award 60NANB6D6206 and the China Scholarship Council (CSC).

References

[1] P. Costa, K.B. Laskey, and K.J. Laskey, PR-OWL: A Bayesian ontology language for the semantic web, *Proc. of Workshop on Uncertainty Reasoning for the Semantic Web (URSW) at the 4th International Semantic Web Conference*, Galway, Ireland, November 2005.

[2] Z. Ding, Y. Peng, A probabilistic extension to ontology language OWL, *Proc. of the 37th Hawaii International Conference on System Sciences*, Big Island, HI, 2004.

[3] Z. Ding, Y. Peng, R. Pan, A Bayesian approach to uncertainty modeling in OWL ontology, *Proc. of 2004 International Conference on Advances in Intelligent Systems - Theory and Applications*, Luxembourg-Kirchberg, Luxembourg, 2004.

[4] Z. Ding, Y. Peng and R. Pan, BayesOWL: Uncertainty modeling in semantic web ontologies, *Soft computing in ontologies and semantic web* (NY: Springer-Verlag, March 2006).

[5] S. Zhang, Y. Sun and Y. Peng, BayesOWL: A prototype system for uncertainty in semantic web, *Proc. of the 2009 International Conference on Artificial Intelligence*, Las Vegas, USA, Jul. 13-16, 2009.

[6] http://www.w3.org/2004/OWL

[7] http://www.w3.org/TR/owl-guide/

[8] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur and Y. Katz, Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics*, 2007, 51-53.

[9] R. Kruithof, Telefoonverkeersrekening, *De Ingenieur 52*, 1937, 15-25.

[10] W.E. Deming and F.F. Stephan, On a least square adjustment of a sampled frequency table when the expected marginal total are known, *Ann. Math. Statist., 11*, 1940, 427-444.

[11] S. Kullback and R.A. Leibler, On information and sufficiency, *Annals of Mathematical Statistics, 22*, 1951, 79-86.

[12] I. Csiszar, I-divergence geometry of probability distributions and minimization problems, *The Annuals of Probability, 3*(1), 1975, 146-158.

[13] Y. Peng and Z. Ding, Modifying Bayesian networks by probability constraints, *Proc. of 21st Conference on Uncertainty in Artificial Intelligence*, Edinburgh, Scotland, July 26-29, 2005.

[14] J. Vomlel, Methods of probabilistic knowledge integration, PhD thesis, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, 1999.

[15] J. Vomlel, Integrating inconsistent data in a probabilistic model, *Journal of Applied NonClassical Logics*, 2003, 1-20.

[16] S. Zhang and Y. Peng, An efficient method for probabilistic knowledge integration, *Proc. of the 20th IEEE International Conference on Tools with Artificial Intelligence*, Dayton, Ohio, Nov. 3-5, 2008.

[17] http://www.norsys.com/

[18] http://clarkparsia.com/pellet

[19] A. Doan, J. Madhavan et al, Learning to match ontologies on the semantic web, *VLDB Journal*, 2003, 303-319.

[20] R. Pan, Z. Ding, Y. Yu and Y. Peng, A Bayesian network approach to ontology mapping, *Proc. of the Fourth International Semantic Web Conference*, Galway, Ireland, November 6-10, 2005.

[21] S. Prasad, Y. Peng, and T. Finin, A tool for mapping between two ontologies using explicit information, Ontologies in Agent Systems (OAS) workshop, *International Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July, 2002.