Schema enrichment in OWL knowledge bases

Tim Schmiedl
Department of Computer Science, University Freiburg
Sundgaualle 52
73110 Freiburg
timschmiedl@neptun.uni-freiburg.de

ABSTRACT
The Semantic Web is steadily growing and the availability of large knowledge graphs increased over the past years. In spite of the growing number of knowledge bases, there exist very few with a sophisticated schema. Often they only consist of a collection of facts with no consistent structure. Other knowledge bases contain only schema information without instances.

But only the combination of both, sophisticated schema and available instance data can enable powerful reasoning, easier consistency checking and improved query abilities.

This article shows two methods for the semantic enrichment of large OWL knowledge bases. The first method focuses at finding and creating class expressions in an automatic or semiautomatic approach, based on the existing data in the graph. Whereas the second method enriches knowledge bases with different types of OWL2 axioms.

Categories and Subject Descriptors
H.2.3 [**Languages**]: Query languages; I.2 [**Learning**]: Concept learning, Induction

General Terms
Theory

Keywords
Ontology engineering, Supervised machine learning, Knowledge Base Enrichment, OWL, Heuristics

1. INTRODUCTION
While the availability and size of knowledge bases has grown exponentially over the last few years, there is often a significantly lack of sophisticated schema. Enrichment of schema information, based on already existing data, is the main subject of this article. To illustrate the benefits of a sophisticated schema together with reliable instance data, consider the following example.

Example 1. Imagine a knowledge base containing famous persons like Barack Obama, Mahatma Gandhi, Elvis Presley etc. and a property called `birthPlace`. An algorithm may find out that every of the person instances contain this property and therefore `birthPlace` can be seen as a functional property with the domain `Person` and the range `Place`.

```
OBjectProperty: birthPlace
  Domain: Person
  Range: Place
  SubPropertyOf: hasBeenAt
```

Adding such an axiom to the knowledge base can have several benefits: (1) Axioms can serve as documentation for the right usage of the schema, (2) adding additional schema can improve consistency and help to debug missing or incorrect information, (3) additional (implicit) information can be inferred. In our example you can observe that every person also `hasBeenAt` their `birthPlace`. Adding new axioms and finding/fixing missing information can be done in a semiautomated approach. A knowledge engineer can decide if new axioms are correct and improve the semantic schema. He than can add the axiom and adjust potentially incorrect or missing data. [4]

In summary, the combination of a solid schema with large instance data allows powerful reasoning, improved query ability and simplified consistency checking.

2. ENRICHMENT OVERVIEW
In this context the term *enrichment* describes the extension of the (semantic) schema of a knowledge base. The process of knowledge base enrichment increases the semantic richness and the expressiveness of an ontology. The goal of the enrichment progress is to find additional axioms, which can be added to an existing ontology. A special case is to find definition of classes and subclasses. This process is closely related to the Inductive Logic Programming (ILP) as described later in this article. Ontology enrichment methods usually depend on machine learning or on applying heuristics to find additional axioms for the knowledge graph. [4]

As stated before, knowledge base enrichment usually works on existing data to improve the semantic schema. It supports the so called *grass-root* approach for creating ontologies. Here the whole ontology structure is not created upfront, but evolves over time and with every part of data that is added to the knowledge base.[4]
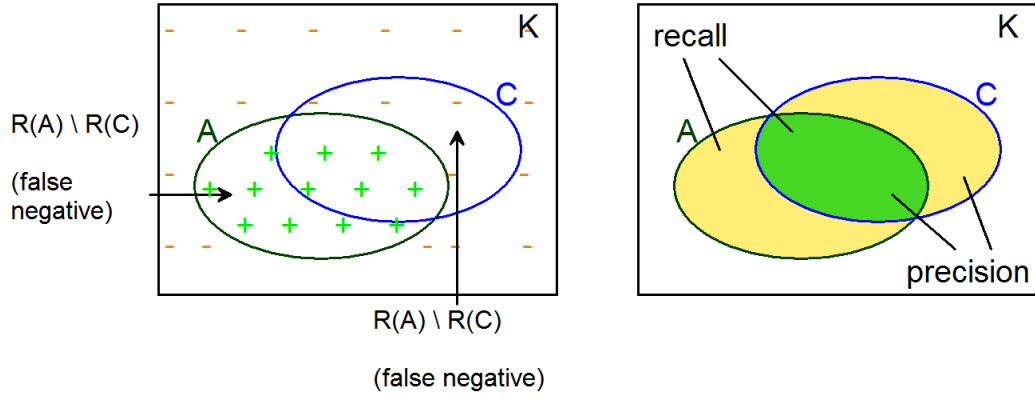
Figure 1: This figure visualises the class A and the expression to test C as a Venn diagram. It also shows the the terms *recall* and *precision*. (Based on the figure in [8])

3. CLASS LEARNING

3.1 Motivation

The class learning approach is one method of enrichment in knowledge bases. It aims at finding new definition of classes to extend the semantic schema. For the motivation of this method examine the following example.

Example 2. For this example consider a knowledge base containing a class `President of the United States` with instance data like Abraham Lincoln, John F. Kennedy, Bill Clinton and Barack Obama. A class Learning algorithm may suggest that the President class is equivalent to the following two class expressions: [1]

1) `Person and born in the USA`
2) `American citizen and born in the USA`

These suggestions would then be presented to a knowledge engineer who than can decide if they are plausible and therefore should be added to the knowledge graph or should be discarded. Should the engineer for instance choose the second statement, he could check if there are instances of the class `President`, where the individual is not also of type American citizen. This could indicate an error or missing information in the knowledge graph. This could then be fixed in this semi-automated approach by the knowledge engineer.

3.2 Learning Problem

The problem of learning class definitions for given data depend on the so called inductive reasoning as opposed to inference or deductive reasoning.[8] Inductive Reasoning is also a key concept in Inductive Logic Programming (ILP).

Definition 1. We are searching for a formal description of the class A, which has existing instances in the examined ontology. A possible class expression C then contains axioms of the form $A \subseteq C$ or $A \equiv C$.

---

[1]The class 'American citizen' is here a subclass of 'Person', which makes the second statement more specific.

This means that the learned expression C is a description of the individuals of A. In our president example, the individuals are the presidents John F. Kennedy, Barack Obama etc. whereas C can be one of the suggested expression. In many cases there will be no exact solution for C, but rather an approximation. This can be the case, if the knowledge base contains false class assignments or missing information. In our example the birthplace of Thomas Jefferson might be missing in the ontology. However, if most of the other presidents have the correct birthplace the learning algorithm may still suggest the expressions. Again, missing information may then be completed by the knowledge engineer.

In a complex knowledge base the class learning algorithm may find many new class definition and often very complicated expression for the same class. Based on Occam's razor [5] simple solutions are preferred over complex ones, because they are more readable an thus easier for the knowledge engineer to evaluate. Simplicity is measured in an straight forward way: the length of an expression, which consists of role, concept and quantifiers. The algorithm presented in [8] is strongly biased towards shorter expressions.

3.3 Algorithm

One algorithm for solving the learning problem is called CELOE (Class Expression Learning for Ontology Engineering). It is described in [8]. A brief overview of CELOE is given in Figure 2. The algorithm follows the "generate and test" approach which is the common concept in ILP.[12] During the learning process many class expression are created and tested against the background knowledge. Each of these class expressions is evaluated using different heuristics which are described in detail in chapter 5.

To find appropriate class expression for describing existing classes, CELOE uses a so called *refinement operator*. The idea of this operator is based on the work in [9],[10] and [11]. Refinement operators are used to search in the space of expressions in a knowlege base. It can be seen as a top-down algorithm and is illustrated in Figure 3. As an example consider the following path ($\rightsquigarrow$ indicates a refinement step):

$T \rightsquigarrow Person \rightsquigarrow Person \sqcap takesPartIn.T \rightsquigarrow Person \sqcap takesPartIn.Meeting$
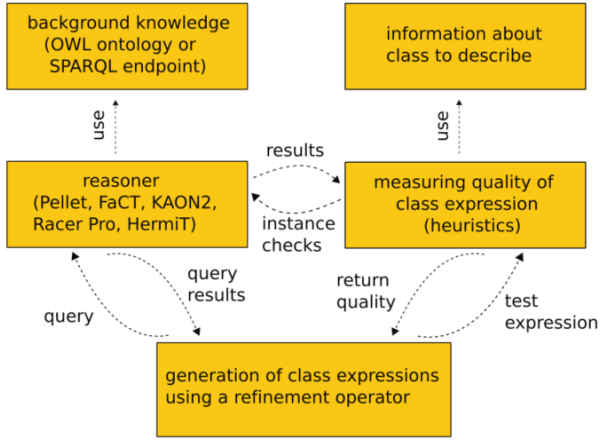
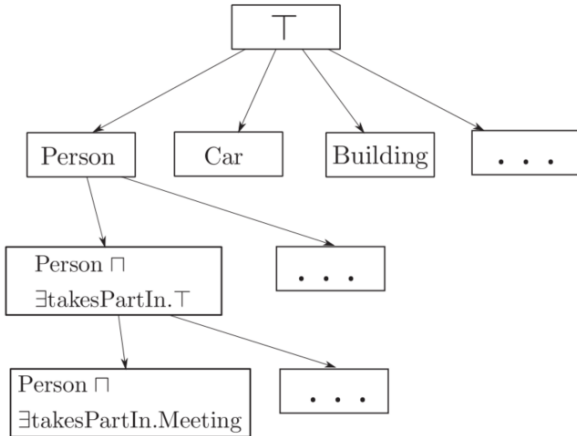Figure 2: Schematic overview of the CELOE algorithm.[8]

Figure 3: A search tree for class expressions is the basis of the refinement operator.[8]
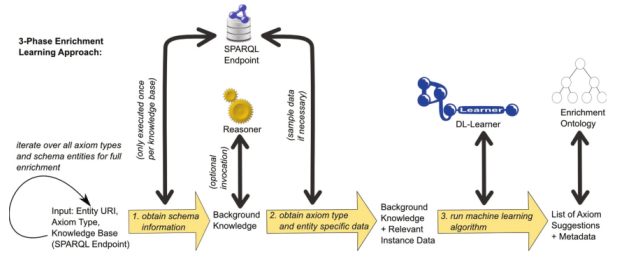

Figure 4: 3-Phase Enrichment Workflow[4]

4. ENRICHMENT WITH OWL AXIOMS

OWL offers many different types of axioms. The method described in [4] uses many different types to enrich the knowledge base. Figure 4 shows the 3 steps in the enrichment workflow:

1. The first phase is about obtaining general information in the knowledge base. In particular axioms, which define the class hierarchy, are obtained. The schema is queried via SPARQL, but have to be loaded only once. The result is stored in memory.

2. In the second phase, data is again obtained via SPARQL. These background checks allows the method to learn and test new axioms. Examples for different axiom types are explained in the following section.

3. The score of the axiom candidates is computed and the result can be evaluated by a knowledge engineer.

The algorithm for suggestion of new axioms is based on checking and counting RDF triples. In the following example we want to learn if the class A is an appropriate domain of a predicate p. For that we count the triples that match this statement to obtain a score.

Example 3. Lets look at the predicate `onto:attendsMeeting` and check if we can find a suitable candidate for a domain. Note that `onto:Manager` is a subclass of `onto:Employee`.

Listing 1: Triples written in turtle syntax

```
onto:Software_Engineer
      onto:attendsMeeting      onto:TeamMeeting;
      rdf:type                 onto:Employee.
onto:Software_Architekt
      onto:attendsMeeting      onto:TeamMeeting;
      rdf:type                 onto:Employee.
onto:Project_Manager
      onto:attendsMeeting      onto:ManagerMeeting;
      rdf:type                 onto:Manager.

onto:Manager    rdfs:subClassOf  onto:Employee.
```

Looking at this example we would obtain a score of 33.3 % (1 out of 3) for the class `onto:Manager` and 100 % (3 out of 3) for the class `onto:Employee`.

Obviously this extreme simple and straightforward method of calculating the score has some limitation. Mainly because

the method doesn't discriminate between a score calculated by having 100 out of 100 correct observations or only 3 out of 3. Different methods, for example the Wald method [1], overcome that problem. More involved heuristics are also shown in the next chapter.

4.1 Obtaining Axioms via SPARQL queries

This section explains how SPARQL queries are used to extract information in the second step of the enrichment workflow.

Subclass and Disjointness of classes

This query evaluates all individuals (`?ind`) and checks if they are instance of the user-defined class definition in `$class`. In this query, a higher value indicates a better candidate for a superclass. Lower values indicate possible disjointness.

```
SELECT ?type (COUNT(?ind ) AS ?count ) WHERE {
    ?ind a <$class >.
    ?ind a ?type .
} GROUP BY ?type
```

Subsumption and Disjointness of properties

A somewhat similar query can be used to learn subsumption and disjointness of *predicates*.

```
SELECT ?p (COUNT(?s ) AS ?count ) WHERE {
    ?s ?p ?o .
    ?s <$property> ?o .
} GROUP BY ?p
```

Domain and Range of properties

A query for the domain of a property counts the occurrences of subjects of type `?type` having the property `$property`.

```
SELECT ?type COUNT(DISTINCT ?ind ) WHERE {
    ?ind <$property> ?o .
    ?ind a ?type .
} GROUP BY ?type
```

The query for the range of `$property` works in a similar way. For properties you can distinguish between object and data properties. Here only the queries for object properties are listed.

```
SELECT ?type (COUNT(DISTINCT ?ind ) AS ?cnt ) WHERE {
    ?s <$property > ? ind .
    ?ind a ?type .
} GROUP BY ?type
```

Inverse of Properties

To check if a property is also inverse, we check the `$property` with subject and object and in swapped position.

```
SELECT ?p (COUNT(*) AS ?count ) WHERE {
    ?s <$property> ?o .
    ?o ?p ?s .
} GROUP BY ?p
```

5. HEURISTICS

5.1 Finding the right heuristic

A heuristic measures how well a given class expression fits the learning problem.[8] To test an algorithm we must have positive and negative examples. As we want to describe class A with the expression C, we can consider every instance of A as a positive and everything else as negative examples. The predictive accuracy can be described as:

$$predacc(C) = 1 - \frac{|R(A)\setminus R(C)|+|R(C)\setminus R(A)|}{n} \quad n = |Ind(K)|$$

Here, Ind(K) stands for the set of individuals in the knowledge base. $R(A) \setminus R(C)$ are the false negatives whereas $R(C) \setminus R(A)$ are the false positives.

As you can see in Figure 1, for the term *precision* we consider the intersection of A and C ($R(A) \cap R(C)$) and the false positive. In other words, how many individuals are rightful considered in our class expression C.

For the other term *recall* we consider again the intersection of A and C and the false negatives. This score tells us how many of the individuals in A we can describe with our class expression C.

Table 1 compares the score of four different heuristics. They are:

- **pred. acc.** The formula for the predictive accuracy was shown above.

- **F-Measure** The F-Measure is defined as the harmonic mean of precision and recall. It can be weighted by a factor β, the authors in [8] choose 3 for β for learning super classes, which gives recall a higher weight than precision.
  $F\text{-}Measure = \frac{\beta+1}{\frac{\beta}{recall}+\frac{1}{precision}}$

- **A-Measure** For the A-Measure we choose the arithmetic mean of precision and recall.
  $A\text{-}Measure = \frac{precision+recall}{2}$

- **Jaccard** The Jaccard index is well known for comparing two sets. It is defined as the size of the intersection divided by the size of the union of the sets.
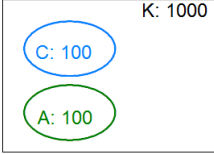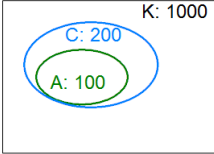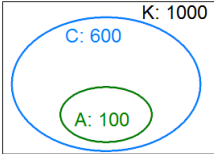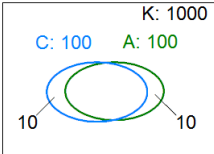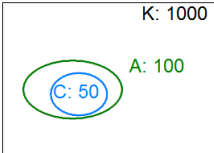 $Jaccard(A, B) = \frac{|A \cap C|}{|A \cup C|}$

5.2 Efficient heuristic computation

For big knowledge base the performance and efficiency of the algorithm is crucial. To compute and test class expression, many retrieval operations are performed. To minimize the cost and improve the performance, the authors in [8] provide three optimisations:

- **Reduction of instance checks** The obvious choice is to reduce the number of instance checks required to test expressions. Assuming that we want to learn an equivalence axiom for class A with the super class A': Instead of checking every individual of the knowledge graph we can restrict our retrieval operation to instances of A'.

- **Approximate and closed world reasoning** To deal with the very high number of instance checks against

Table 1: Various examples to illustrate the scores of different heuristics.

Illustration	accuracy & recall		pred. acc.	F-Measure	A-Measure	Jaccard
K: 1000, C: 100, A: 100	accuracy recall	0% 0%	80 %	0 %	0 %	0 %
K: 1000, C: 200, A: 100	accuracy recall	50% 100%	90 %	66.7 %	75 %	50 %
K: 1000, C: 600, A: 100	accuracy recall	16.7% 100%	50 %	28.6 %	58.3 %	16.7 %
K: 1000, C: 100, A: 100, 10, 10	accuracy recall	90% 90%	98 %	90 %	90 %	81.8 %
K: 1000, A: 100, C: 50	accuracy recall	50% 100%	95 %	66.7 %	75 %	50 %

the knowledge base, CELOE uses a special reasoner.[8] It uses an approximate and incomplete reasoning for fast instance checks (FIC). The reasoner depends on the first run on a standard OWL reasoner to check instances and property relationships. The result of this first reasoning is stored in memory for fast (but approximated) instance checks. The reasoner follows a closed world assumption.

- **Stochastic coverage computation** This optimisation again reduces the necessary instance checks. Looking at the different heuristics, we can see that $|R(A)|$ needs to be computed only once whereas e.g. the expensive expression $R(A) \cap R(C)$ is computed for every different C. To improve performance we can try to approximate the result by testing randomly drawn objects and checking if we are sufficiently confident that our estimation is within a certain bound.

6. EVALUATION

To evaluate the methods of class learning, the authors of [8] tested their algorithm on a variety of real world ontologies of different sizes and domains. The goal of the evaluation consisted of 3 parts: determine the influence of reasoning and heuristics on suggestions, test the performance and efficiency on large real world ontologies and to test the accuracy of approximations described in section 5.2.

To perform the evaluation, the authors in [8] wrote a dedicated plugin for the Protégé editor. The plugin first looks for classes with enough instances. For each of these classes they ran the CELOE algorithm to generate suggestions for definitions. Here they tested two different reasoner and the previously described heuristics. The results of these tests were categorized in three main categories: (1) the suggestion improves the ontology (improvement), (2) the suggestion does not improve the ontology and therefore should not be added to the ontology (not acceptable), (3) adding the suggestion results in a modelling error (error). A small part of the results is shown in Table 2, the full evaluation results are shown in [8].

The second evaluation results were based on the performance of the approximation reasoner as described in section 5.2. The tests showed that an approximation lead to significant performance improvement, this is especially the case for larger ontologies. The time required to test a class expression showed smaller variations and a performance gain of several orders of magnitudes has been achieved. The approximation has shown to be very accurate and had hardly any influence on the learning algorithm.

Table 2: Evaluation results of [8]. Accuracy of different heuristics.

Reasoner/heuristic	Improv.	Not acc.	missed Improv.
Pellet/F-Measure	16.70	64.66	14.95
Pellet/A-Measure	16.70	64.66	14.95
Pellet/pred.acc.	16.59	64.93	15.22
Pellet FIC/F-Measure	36.60	52.62	1.90
Pellet FIC/A-Measure	36.19	52.84	1.63
Pellet FIC/pred.acc.	32.99	52.58	4.35

Preliminary Evaluation has also been done in [4] for OWL axiom enrichment. The authors choose DBpedia to test their algorithms. Table 3 shows a small subset of the results. In [4] the evaluation is discussed in great detail. For example the three newly discovered symmetric object properties in the DBpedia ontology were: `dbo:neighboringMunicipaly`, `dbo:sisterCollege` and `dbo:currentPartner`.

Table 3: Evaluation results of [4] (combined object and data properties)

axiom type	recall	new axioms	precision
SubClassOf	180/185	155	75 %
EquivalentClasses	0/0	1812	50 %
DisjointClasses	0/0	2449	100 %
PropertyDomain	833/942	1298	54 %
PropertyRange	291/1032	500	46 %
SymmetricProperty	0/0	3	100 %

7. RELATED WORK

Related work can be divided into two categories: the first part covers supervised machine learning with OWL, the second part is focused on (semi-)automated ontology engineering methods.

Early work of supervised learning in description logic was published in [6, 7], which uses the so called *least common subsumer* to solve the learning problem. Later work invented the concept of the refinement operator to solve the problem in a top-down approach.[3] The refinement operator was later adapted for description logic [9, 10, 11] and is also used in CELOE, as mentioned before.

The starting point of (semi-)automatic ontology engineering was set by [13], a formal concept analysis was described in [2]. Another interesting approach is presented in [14] which proposes to improve knowledge bases through relation exploration. It is implemented in the RELExO framework.

8. CONCLUSIONS

The algorithms and methods summarized in this article, like they were presented in [8] and [4], show promising results. The authors of [8] analysed their CELOE algorithm with different heuristics and proved that approximating heuristic values are often very accurate while improving the performance significantly. The strong bias towards shorter expressions increase the readability of suggestions and help the knowledge engineer to verify additional statements.

The enrichment methods in [4] also show great potential. The evaluation of this approach showed that the method also works on very large scales, like the size of the DBpedia knowledge base. In future the authors will investigate enhancements in the presented methods and collaborate with the authors of [15] to further fine-tune the approach.

9. REFERENCES

[1] A. Agresti and B. A. Coull. Approximate is better than ŞexactŤ for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.

[2] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing description logic knowledge bases using formal concept analysis. In *In Proc. of IJCAI 2007*, pages 230–235. AAAI Press, 2007.

[3] L. Badea and S.-H. Nienhuys-Cheng. A refinement operator for description logics. In *Inductive logic programming*, pages 40–59. Springer, 2000.

[4] L. Bühmann and J. Lehmann. Universal owl axiom enrichment for large knowledge bases. *EKAW*, pages 57–71, 2012.

[5] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occams's razor. In *In Readings in Machine Learning*, pages 201–204.

[6] W. W. Cohen, A. Borgida, and H. Hirsh. Computing least common subsumers in description logics. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 754–760. AAAI Press, 1992.

[7] W. W. Cohen and H. Hirsh. Learning the classic description logic: Theoretical and experimental results. *KR*, 94:121–133, 1994.

[8] J. Lehmann, S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, pages 71–81, 2011.

[9] J. Lehmann and P. Hitzler. Foundations of refinement operators for description logics. *LNCS*, 4894, 2007.

[10] J. Lehmann and P. Hitzler. A refinement operator based learning algorithm for the alc description logic. *LNCS*, 4894, 2007.

[11] J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning Journal*, 78:203–250, 2010.

[12] S.-H. Nienhuys-Cheng and R. d. Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[13] S. Rudolph. Exploring relational structures via fle. In *Conceptual Structures at Work: 12th International Conference on Conceptual Structures. Volume 3127 of LNCS*. Springer, 2004.

[14] J. Völker and S. Rudolph. Fostering web intelligence by semi-automatic owl ontology refinement. IEEE, 2008.

[15] J. Völker and M. Niepert. Statistical schema induction. In *The Semantic Web: Research and Applications*, pages 124–138. Springer, 2011.