Class Expression Learning for Ontology Engineering

Jens Lehmann *, Sören Auer, Lorenz Bühmann, Sebastian Tramp

Universität Leipzig, Department of Computer Science, Johannisgasse 26, D-04103 Leipzig, Germany

Abstract

While the number of knowledge bases in the Semantic Web increases, the maintenance and creation of ontology schemata still remain a challenge. In particular creating class expressions constitutes one of the more demanding aspects of ontology engineering. In this article we describe how to adapt a semi-automatic method for learning OWL class expressions to the ontology engineering use case. Specifically, we describe how to extend an existing learning algorithm for the class learning problem. We perform rigorous performance optimization of the underlying algorithms for providing instant suggestions to the user. We also present two plugins, which use the algorithm, for the popular Protégé and OntoWiki ontology editors and provide a preliminary evaluation on real ontologies.

Key words: Ontology Engineering, Supervised Machine Learning, Concept Learning, Ontology Editor Plugins, OWL, Heuristics

1 Introduction and Motivation

The Semantic Web has recently seen a rise in the availability and usage of knowledge bases, as can be observed within the Linking Open Data Initiative, the TONES and Protégé ontology repositories, or the Watson search engine. Despite this growth, there is still a lack of knowledge bases that consist of sophisticated schema information and instance data adhering to this schema.

* Corresponding author.
Email addresses: lehmann@informatik.uni-leipzig.de (Jens Lehmann), auer@informatik.uni-leipzig.de (Sören Auer), buehmann@informatik.uni-leipzig.de (Lorenz Bühmann), tramp@informatik.uni-leipzig.de (Sebastian Tramp).

Several knowledge bases, e.g. in the life sciences, only consist of schema information, while others are, to a large extent, a collection of facts without a clear structure, e.g. information extracted from data bases or texts. The combination of sophisticated schema and instance data allows powerful reasoning, consistency checking, and improved querying possibilities. We argue that being able to learn OWL class expressions [1] is a step towards achieving this goal.

Example 1 As an example, consider a knowledge base containing a class `Capital` and instances of this class, e.g. London, Paris, Washington, Canberra etc. A machine learning algorithm could, then, suggest that the class `Capital` may be equivalent to one of the following OWL class expressions in Manchester OWL syntax [2]:

```
City and isCapitalOf at least one GeopoliticalRegion
City and isCapitalOf at least one Country
```

Both suggestions could be plausible: The first one is more general and includes cities that are capitals of states, whereas the latter one is stricter and limits the instances to capitals of countries. A knowledge engineer can decide which one is more appropriate, i.e. a semi-automatic approach is used, and the machine learning algorithm should guide her by pointing out which one fits the existing instances better. Assuming the knowledge engineer decides for the latter, an algorithm can show her whether there are instances of the class `Capital` which are neither instances of `City` nor related via the property `isCapitalOf` to an instance of `Country`. [3] The knowledge engineer can then continue to look at those instances and assign them to a different class as well as provide more complete information; thus improving the quality of the knowledge base. After adding the definition of `Capital`, an OWL reasoner can compute further instances of the class which have not been explicitly assigned before.

We argue that the approach and plugins presented here are the first ones to be practically usable by knowledge engineers for learning class expressions. Using machine learning for the generation of suggestions instead of entering them manually has the advantage that 1.) the given suggestions fit the instance data, i.e. schema and instances are developed in concordance, and 2.) the entrance barrier for knowledge engineers is significantly lower, since understanding an OWL class expression is easier than analysing the structure of the knowledge base and creating a class expression manually. Disadvantages of the approach are the dependency on the availability of instance data in the knowledge base and requirements on the quality of the ontology, i.e. modelling errors in the ontology can reduce the quality of results.

[1] `http://www.w3.org/TR/owl2-syntax/#Class_Expressions`
[2] For details on Manchester OWL syntax (e.g. used in Protégé, OntoWiki) see [18].
[3] This is not an inconsistency under the standard OWL open world assumption, but rather a hint towards a potential modelling error.

Overall, we make the following contributions:

- extension of an existing learning algorithm for learning class expressions to the ontology engineering scenario,
- evaluation of five different heuristics,
- rigorous performance improvements including an instance check method for class expression learning and a stochastic coverage approximation method,
- showcase how the enhanced ontology engineering process can be supported with plugins for Protégé and OntoWiki,
- evaluation with several real ontologies from various domains.

The adapted algorithm for solving the learning problems, which occur in the ontology engineering process, is called *CELOE (Class Expression Learning for Ontology Engineering)*. We have implemented the algorithm within the open-source framework DL-Learner. [4] DL-Learner [22,23] leverages a modular architecture, which allows to define different types of components: knowledge sources (e.g. OWL files), reasoners (e.g. DIG [5] or OWL API based), learning problems, and learning algorithms. In this article we focus on the latter two component types, i.e. we define the class expression learning problem in ontology engineering and provide an algorithm for solving it.

The paper is structured as follows: Section 2 covers basic notions and Section 3 describes a heuristic for class expression learning in ontology engineering. Thereafter, Section 4 shows how this heuristic can be computed efficiently. Section 5 briefly explains changes compared to previous algorithms. We then describe the implemented plugins in Section 6 and 7 for Protégé and OntoWiki, respectively. The algorithm is evaluated in Section 8. Finally, in Section 9 and 10 we conclude with related and future work.

2 Preliminaries

For an introduction to OWL and description logics, we refer to [4] and [17].

2.1 Learning Problem

The process of learning in logics, i.e. trying to find high-level explanations for given data, is also called *inductive reasoning* as opposed to *inference* or *deductive reasoning*. The main difference is that in deductive reasoning it is formally shown whether a statement follows from a knowledge base, whereas in inductive learning new statements are invented. Learning problems, which

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

are similar to the one we will analyse, have been investigated in *Inductive Logic Programming* [29] and, in fact, the method presented here can be used to solve a variety of machine learning tasks apart from ontology engineering.

In the ontology learning problem we consider, we want to learn a formal description of a class A, which has (inferred or asserted) instances in the considered ontology. In the case that A is already described by a class expression C via axioms of the form $A \sqsubseteq C$ or $A \equiv C$, those can be either refined, i.e. specialised/generalised, or relearned from scratch by the learning algorithm. To define the class learning problem, we need the notion of a *retrieval* reasoner operation $R_{\mathcal{K}}(C)$. $R_{\mathcal{K}}(C)$ returns the set of all instances of C in a knowledge base $\mathcal{K}$. If $\mathcal{K}$ is clear from the context, the subscript can be ommitted.

Definition 1 (class learning problem) Let an existing named class A in a knowledge base $\mathcal{K}$ be given. The *class learning problem* is to find an expression C such that $R_{\mathcal{K}}(C) = R_{\mathcal{K}}(A)$.

Clearly, the learned expression C is a description of (the instances of) A. Such an expression is a candidate for adding an axiom of the form $A \equiv C$ or $A \sqsubseteq C$ to the knowledge base $\mathcal{K}$. If a solution of the learning problem exists, then the used base learning algorithm (as presented in the following subsection) is complete, i.e. guaranteed to find a correct solution if one exists in the target language and there are no time and memory constraints (see [25,26] for the proof). In most cases, we will not find a solution to the learning problem, but rather an approximation. This is natural, since a knowledge base may contain false class assignments or some objects in the knowledge base are described at different levels of detail. For instance, in Example 1, the city "Apia" might be typed as "Capital" in a knowledge base, but not related to the country "Samoa". However, if most of the other cities are related to countries via a role `isCapitalOf`, then the learning algorithm may still suggest `City and isCapitalOf at least one Country` since this describes the majority of capitals in the knowledge base well. If the knowledge engineer agrees with such a definition, then a tool can assist him in completing missing information about some capitals.

By Occam's razor [8] simple solutions of the learning problem are to be preferred over more complex ones, because they are more readable. This is even more important in the ontology engineering context, where it is essential to suggest simple expressions to the knowledge engineer. We measure simplicity as the *length* of an expression, which is defined in a straightforward way, namely as the sum of the numbers of concept, role, quantifier, and connective symbols occurring in the expression. The algorithm is biased towards shorter expressions. Also note that, for simplicity the definition of the learning problem itself does enforce coverage, but not prediction, i.e. correct classification of objects which are added to the knowledge base in the future. Concepts with high coverage and poor prediction are said to *overfit* the data. However, due

to the strong bias towards short expressions this problem occurs empirically rare in description logics [26].

2.2 Base Learning Algorithm

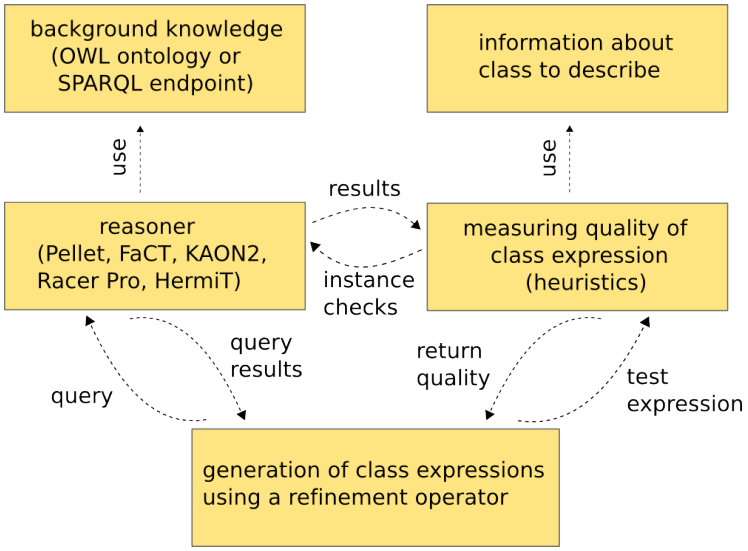


Figure 1. Outline of the general learning approach in CELOE: One part of the algorithm is the generation of promising class expressions taking the available background knowledge into account. Another part is a heuristic measure of how close an expression is to being a solution of the learning problem. Figure adapted from [16].

Figure 1 gives a brief overview of our algorithm *CELOE*, which follows the common "generate and test" approach in ILP. This means that learning is seen as a search process and several class expressions are generated and tested against a background knowledge base. Each of those class expressions is evaluated using a heuristic, which is described in the next section. A challenging part of a learning algorithm is to decide which expressions to test. In particular, such a decision should take the computed heuristic values and the structure of the background knowledge into account. For CELOE, we use the approach described in [25,26] as base, where this problem has already been analysed, implemented, and evaluated in depth. It is based on the idea of *refinement operators*:

Definition 2 (refinement operator) A *quasi-ordering* is a reflexive and transitive relation. In a quasi-ordered space $(S, \preceq)$ a *downward (upward) refinement operator* ρ is a mapping from S to 2^S, such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ ($C \preceq C'$). C' is called a *specialisation* (*generalisation*) of C.

Refinement operators can be used for searching in the space of expressions. As ordering we can use subsumption. (Note that the subsumption relation $\sqsubseteq$ is a quasi-ordering.) If an expression C subsumes an expression D ($D \sqsubseteq C$), then C will cover all examples which are covered by D. This makes subsumption a suitable order for searching in expressions as it allows to prune parts of the search space without losing possible solutions.

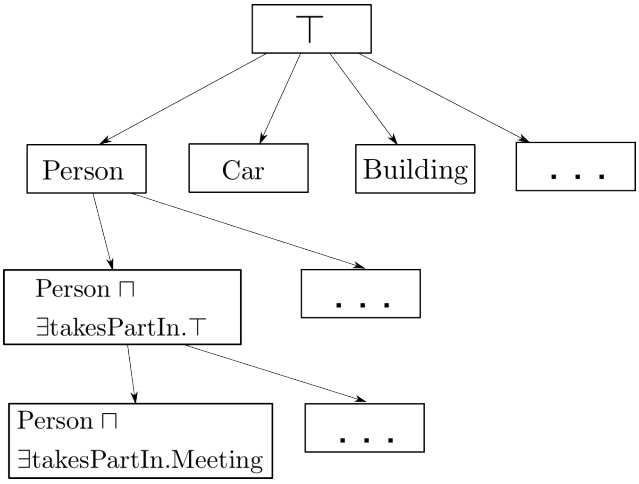


Figure 2. Illustration of a search tree in a top down refinement approach.

The approach we used is a top-down algorithm based on refinement operators as illustrated in Figure 2. This means that the first class expression, which will be tested is the most general expression ($\top$), which is then mapped to a set of more specific expressions by means of a downward refinement operator. Naturally, the refinement operator can be applied to the obtained expressions again, thereby spanning a *search tree*. The search tree can be pruned when an expression does not cover sufficiently many instances of the class A we want to describe. One example for a path in a search tree spanned up by a downward refinement operator is the following ($\rightsquigarrow$ denotes a refinement step):

$$\top \rightsquigarrow \texttt{Person} \rightsquigarrow \texttt{Person} \sqcap \texttt{takesPartinIn}.\top \rightsquigarrow \texttt{Person} \sqcap \texttt{takesPartIn.Meeting}$$

The heart of such a learning strategy is to define a suitable refinement operator and an appropriate search heuristics for deciding which nodes in the search tree should be expanded. The refinement operator in the considered algorithm is defined in [26]. It is based on earlier work in [25] which in turn is build on theoretical foundations in [24]. It has been shown to be the best achievable operator with respect to a set of properties (not further described here), which are used to assess the performance of refinement operators. The learning algorithm supports conjunction, disjunction, negation, existential and universal quantifiers, cardinality restrictions, hasValue restrictions as well as boolean and double datatypes.

3 Finding a Suitable Heuristic

A heuristic measures how well a given class expression fits a learning problem and is used to guide the search in a learning process. To define a suitable

heuristic, we first need to address the question of how to measure the accuracy of a class expression. We introduce several heuristics, which can be used for *CELOE* and later evaluate them.

We cannot simply use supervised learning from examples right-away, since we do not have positive and negative examples available. We can try to tackle this problem by using the existing instances of the class as positive examples and the remaining instances as negative examples. This is illustrated in Figure 3, where $\mathcal{K}$ stands for the knowledge base and A for the class to describe. We can then measure accuracy as the number of correctly classified examples divided by the number of all examples. This can be computed as follows for a class expression C and is known as *predictive accuracy* in Machine Learning:

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

Here, $Ind(\mathcal{K})$ stands for the set of individuals occurring in the knowledge base. $R(A) \setminus R(C)$ are the false negatives whereas $R(C) \setminus R(A)$ are false positives. n is the number of all examples, which is equal to the number of individuals in the knowledge base in this case. Apart from learning definitions, we also want to be able to learn super class axioms ($A \sqsubseteq C$). Naturally, in this scenario $R(C)$ should be a superset of $R(A)$. However, we still do want $R(C)$ to be as small as possible, otherwise $\top$ would always be a solution. To reflect this in our accuracy computation, we penalise false negatives more than false positives by a factor of t ($t > 1$) and map the result to the interval $[0, 1]$:

$$predacc(C, t) = 1 - 2 \cdot \frac{t \cdot |R(A) \setminus R(C)| + |R(C) \setminus R(A)|}{(t+1) \cdot n} \quad n = |Ind(\mathcal{K})|$$

While being straightforward, the outlined approach of casting class learning into a standard learning problem with positive and negative examples has the disadvantage that the number of negative examples will usually be much higher than the number of positive examples. As shown in Table 1, this may lead to overly optimistic estimates. More importantly, this accuracy measure has the drawback of having a dependency on the number of instances in the knowledge base.

Therefore, we investigated further heuristics, which overcome this problem, in particular by transferring common heuristics from information retrieval to the class learning problem:

(1) *F-Measure:* F_β-Measure is based on *precision* and *recall* weighted by β. They can be computed for the class learning problem without having negative examples. Instead, we perform a retrieval for the expression C, which we want to evaluate. We can then define precision as the percentage of instances of C, which are also instances of A and recall as percentage of instances of A, which are also instances of C. This is visualised in

Figure 3. F-Measure is defined as harmonic mean of precision and recall. For learning super classes, we use F_3 measure by default, which gives recall a higher weight than precision.

(2) *A-Measure:* We denote the arithmetic mean of precision and recall as A-Measure. Super class learning is achieved by assigning a higher weight to recall. Using the arithmetic mean of precision and recall is uncommon in Machine Learning, since it results in too optimistic estimates. However, we found that it is useful in super class learning, where F_n is often too pessimistic even for higher n.

(3) *Generalised F-Measure:* Generalised F-Measure has been published in [12] and extends the idea of F-measure by taking the three valued nature of classification in OWL/DLs into account: An individual can either belong to a class, the negation of a class or none of both cases can be proven. This differs from common binary classification tasks and, therefore, appropriate measures have been introduced (see [12] for details). Adaption for super class learning can be done in a similar fashion as for F-Measure itself.

(4) *Jaccard Distance:* Since $R(A)$ and $R(C)$ are sets, we can use the well-known Jaccard coefficient to measure the similarity between both sets.

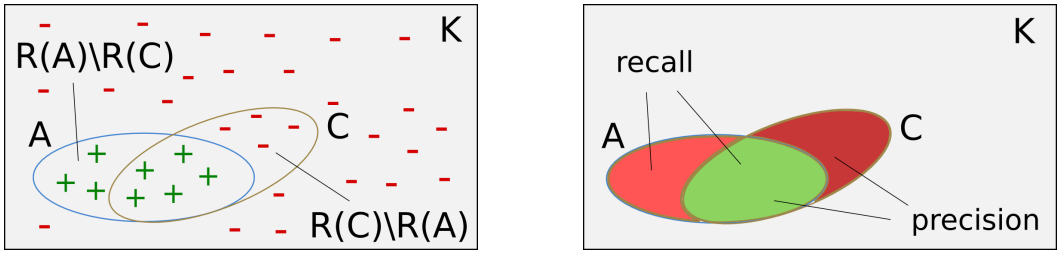


Figure 3. Visualisation of different accuracy measurement approaches. $\mathcal{K}$ is the knowledge base, A the class to describe and C a class expression to be tested. Left side: Standard supervised approach based on using positive (instances of A) and negative (remaining instances) examples. Here, the accuracy of C depends on the number of individuals in the knowledge base. Right side: Evaluation based on two criteria: recall (*Which fraction of* $R(A)$ *is in* $R(C)$*?*) and precision (*Which fraction of* $R(C)$ *is in* $R(A)$*?*).

We argue that those four measures are more appropriate than predictive accuracy when applying standard learning algorithms to the ontology engineering use case. Table 1 provides some example calculations, which allow the reader to compare the different heuristics.

4 Efficient Heuristic Computation

Most of the runtime of a learning algorithm is spent for computing heuristic values, since they require expensive reasoner requests. In particular, retrieval operations are often required. Performing an instance retrieval can be very

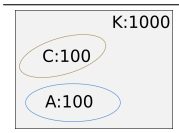
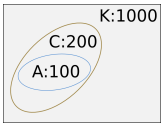
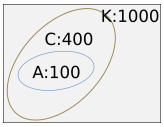
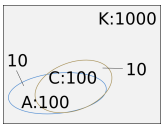
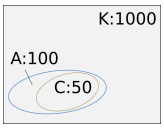
illustration	pred. acc.		F-Measure		A-Measure		Jaccard
	eq	sc	eq	sc	eq	sc	
	80%	67%	0%	0%	0%	0%	0%
	90%	92%	67%	73%	75%	88%	50%
	70%	75%	40%	48%	63%	82%	25%
	98%	97%	90%	90%	90%	90%	82%
	95%	88%	67%	61%	75%	63%	50%

Table 1
Example accuracies for selected cases (eq = equivalence class axiom, sc = super class axiom). The images on the left represent an imaginary knowledge base $\mathcal{K}$ with 1000 individuals, where we want to describe the class A by using expression C. It is apparent that using predictive accuracy leads to impractical accuracies, e.g. in the first row C cannot possibly be a good description of A, but we still get 80% accuracy, since all the negative examples outside of A and C are correctly classified.

expensive for large knowledge bases. Depending on the ontology schema, this may require instance checks for many or even all objects in the knowledge base. Furthermore, a machine learning algorithm easily needs to compute the score for thousands of expressions due to the expressiveness of the target language OWL. We provide three performance optimisations:

Reduction of Instance Checks The first optimisation is to reduce the number of objects we are looking at by using background knowledge. Assuming that we want to learn an equivalence axiom for class A with super class A', we can start a top-down search in our learning algorithm with A' instead of $\top$. Thus, we know that each expression we test is a subclass of A', which allows us to restrict the retrieval operation to instances of A'. This can be generalised to several super classes and a similar observation applies to learning super class axioms.

Approximate and Closed World Reasoning The second optimisation is to use a reasoner designed for performing a very high number of instance checks against a knowledge base which can be considered static, i.e. we assume it is not changed during the run of the algorithm. This is reasonable, since

the algorithm runtimes are usually in the range of a few seconds. In CELOE, we use an own approximate incomplete reasoning procedure for fast instance checks (FIC) which partially follows a closed world assumption (CWA). First, we use a standard OWL reasoner like Pellet to compute the instances of named classes occurring in the learning process as well as the property relationships. Afterwards, the FIC procedure can answer all instance checks (approximately) by using only the inferred knowledge, which results in an order of magnitude speedup compared to using a standard reasoner for instance checks as we will show in Section 8. The second step, i.e. the instance checks from the inferred knowledge in memory, follows a closed world assumption.

We briefly want to discuss why we are preferring the CWA over a straightforward use of a standard OWL reasoner. Note that this has been explained in [7] already, but we repeat the argument here. Consider the following knowledge base containing a person a with two male children a_1 and a_2:

$$\begin{aligned}\mathcal{K} = \{&\texttt{Male} \sqsubseteq \texttt{Person},\\ &\texttt{OnlyMaleChildren}(a),\\ &\texttt{Person}(a), \texttt{Male}(a_1), \texttt{Male}(a_2),\\ &\texttt{hasChild}(a, a_1), \texttt{hasChild}(a, a_2)\}\end{aligned}$$

Assume, we want to learn a description for the named class `OnlyMaleChildren`. If we want to compute the score for expression $C = \texttt{Person} \sqcap \forall \texttt{hasChild}.\texttt{Male}$, describing persons with only male children, we need to check whether a is an instance of C. It turns out that this is true under CWA and not true under OWA. However, C is a good description of a and could be used to define `OnlyMaleChildren`. For this reason, CWA is usually preferred in this Machine Learning scenario. Otherwise, universal quantifiers and number restrictions would hardly occur (unnegated) in suggestions for the knowledge engineer – even if their use would be perfectly reasonable. In a broader view, the task of the learning algorithm is to inspect the data actually present, whereas the knowledge engineer can then decide whether these observations hold in general.

Stochastic Coverage Computation The third optimisation is a further reduction of necessary instance checks. Looking at the various heuristics in detail, we can observe that $|R(A)|$ needs to be computed only once, while other expressions like $|R(A) \cap R(C)|$ (number of common instances of A and C) or, particularly, $|R(C)|$ are expensive to compute. However, since the heuristic provides only a rough guidance for the learning algorithm, it is usually sufficient to approximate it and, thus, to test more class expressions within a certain time span. However, an all too inaccurate approximation can also increase the number of expressions to test, since the algorithm is more likely to consider less relevant areas of the search space.

The approximation works by testing randomly drawn objects and terminating when we are sufficiently confident that our estimation is within a δ-bound. This can be done for all heuristics. As an example, we want to illustrate the computation of F-Measure for the simple case that we approximate $|R(C)|$ (i.e. we do not consider the more involved case that $|R(A) \cap R(C)|$ should also approximated in this example).

Using the equivalent expression $|R(A) \cap R(C)| + |R(C) \setminus R(A)|$ for $|R(C)|$ and replacing $|R(C) \setminus R(A)|$ with x in the formula for F-Measure, we get:

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \tag{1}$$

$$= (1+\beta^2) \cdot \frac{\frac{|R(A)\cap R(C)|}{|R(A)\cap R(C)|+x} \cdot \frac{|R(A)\cap R(C)|}{|(R(A)|}}{\beta^2 \cdot \frac{|R(A)\cap R(C)|}{|R(A)\cap R(C)|+x} + \frac{|R(A)\cap R(C)|}{|(R(A)|}} \tag{2}$$

Given that $|R(A)|$ and $|R(A) \cap R(C)|$ are already known, we can look at this as a function of x, where x is the number of instances of C which are not instances of A. It is monotonically decreasing for positive values of x, because lower values of x mean lower precision and, thus, lower F-Measure. We now approximate x by drawing random individuals from the class A' (explained in the paragraph about reduction of instance checks). From those results, we can compute a confidence interval efficiently by using the improved Wald method defined in [2]. Assume that we have performed m instance checks where s of them were successful (true), then the 95% confidence interval is as follows:

$$\max(0, p' - 1.96 \cdot \sqrt{\frac{p' \cdot (1-p')}{m+4}}) \text{ to } \min(1, p' + 1.96 \cdot \sqrt{\frac{p' \cdot (1-p')}{m+4}})$$

$$\text{with } p' = \frac{s+2}{m+4}$$

This formula is easy to compute and has been shown to be accurate in [2]. Let x_1 and x_2 be the lower and upper border of the confidence interval. We draw instances of A until the interval is smaller than a configurable maximum width δ:

$$\delta \geq F_\beta(x_1) - F_\beta(x_2)$$

Since $F_\beta(x)$ is monotonic, the difference $F_\beta(x_1) - F_\beta(x_2)$ is the maximum difference between two function values in the interval $[x_1, x_2]$. If this value is smaller than δ, we can be confident that we are within a δ range of the real value of F_β. The choice of δ depends on the desired accuracy. For CELOE, we use $\delta = 0.05$, because of empirical studies (see also Table 4).

This method can be applied to most heuristics in a similar fashion. In some cases, two variables, e.g. those estimating $|R(A) \cap R(C)|$ and $|R(A)|$, can be

approximated within one heuristic, if it can be shown that the corresponding δ range estimate is pessimistic. Intuitively, the reason is that it is unlikely that both real values of the approximated variables are outside of their computed confidence interval. Details and an approximation for other another heuristic can be found in [23]. Note that it is usually not hard to show that the approximation process always terminates.

Example 2 Assume we want to learn an equivalence axiom, i.e. $\beta = 1$, for a class A, where A has 1,000 instances and the super classes of A, excluding A itself, have 10,000 instances. We choose F-Measure as heuristic. Let C be an expression to test and $|R(A) \cap R(C)| = 800$. We use $\delta = 0,05$ as approximation parameter.

We now start drawing random instances of super classes of A excluding A itself, i.e. we pick amongst 10,000 individuals and perform instance checks. Assume, we have checked 41 individuals out of which 31 were instances of C. According to the Wald method, the 95% confidence interval multiplied by 10,000 ranges from $x_1 = 6049$ to $x_2 = 8635$. Thus, $F_1(x_1) - F_1(x_2) = 0.0505$ according to Equation 1. Since this value is larger than δ, we need to perform another instance check. Assuming this instance check is positive, we get a new 95% confidence interval ranging from 6130 to 8669 and $F_1(x_1) - F_1(x_2)$ is now 0.0489. At this point, the approximation process terminates and we obtain an F_1 score of 0.1699. To approximate the heuristic value, we have only checked 1 042 instead of 11 000 individuals.

In general, the performance gain through this method is higher for larger knowledge bases (more specifically large Aboxes). There is, however, almost no performance loss for smaller knowledge bases, because of the efficient Wald method. It is noteworthy that the method has impact on a number of other scenarios, where heuristic values need only be approximated. For instance, a recent article [35] pointed out that most ILP methods do not scale to a high number of examples. Using the technique introduced above, such problems can indeed be handled.

5 Adaptation of the Learning Algorithm

While the major change compared to other supervised learning algorithms for OWL is the previously described heuristic, we also made further modifications. The goal of those changes is to adapt the learning algorithm to the ontology engineering scenario: For example, the algorithm was modified in order to introduce a strong bias towards short class expressions. This means that the algorithm is less likely to produce long class expressions, but is almost guaranteed to find any suitable short expression. Clearly, the rationale behind this

change is that knowledge engineers can understand short expressions better than more complex ones and it is essential not to miss those.

We also introduced improvements to enhance the readability of suggestions: Each suggestion is reduced, i.e. there is a guarantee that they are as succinct as possible. For example, $\exists \texttt{hasLeader}.\top \sqcap \texttt{Capital}$ is reduced to `Capital` if the background knowledge allows to infer that a capital is a city and each city has a leader. This reduction algorithm uses the complete and sound *Pellet* reasoner, i.e. it can take any possible complex relationships into account by performing a series of subsumption checks between class expressions. A caching mechanism is used to store the results of those checks, which allows to perform the reduction very efficiently after a warm-up phase.

Furthermore, we also make sure that "redundant" suggestions are omitted. If one suggestion is longer and subsumed by another suggestion and both have the same characteristics, i.e. classify the relevant individuals equally, the more specific suggestion is filtered. This avoids expressions containing irrelevant subexpressions and ensures that the suggestions are sufficiently diverse.

6 The Protégé Plugin

After implementing and testing the described learning algorithm, we integrated it into *Protégé* and *OntoWiki*. Together with the Protégé developers, we extended the Protégé 4 plugin mechanism to be able to seamlessly integrate the DL-Learner plugin as an additional method to create class expressions. This means that the knowledge engineer can use the algorithm exactly where it is needed without any additional configuration steps. The plugin has also become part of the official Protégé 4 repository, i.e. it can be directly installed from within Protégé.

A screenshot of the plugin is shown in Figure 4. To use the plugin, the knowledge engineer is only required to press a button, which then starts a new thread in the background. This thread executes the learning algorithm. The used algorithm is an *anytime algorithm*, i.e. at each point in time we can always see the currently best suggestions. The GUI updates the suggestion list each second until the maximum runtime – 10 seconds per default – is reached. This means that the perceived runtime, i.e. the time after which only minor updates occur in the suggestion list, is often only one or two seconds for small ontologies. For each suggestion, the plugin displays its accuracy.

When clicking on a suggestion, it is visualized by displaying two circles: One stands for the instances of the class to describe and another circle for the instances of the suggested class expression. Ideally, both circles overlap com-

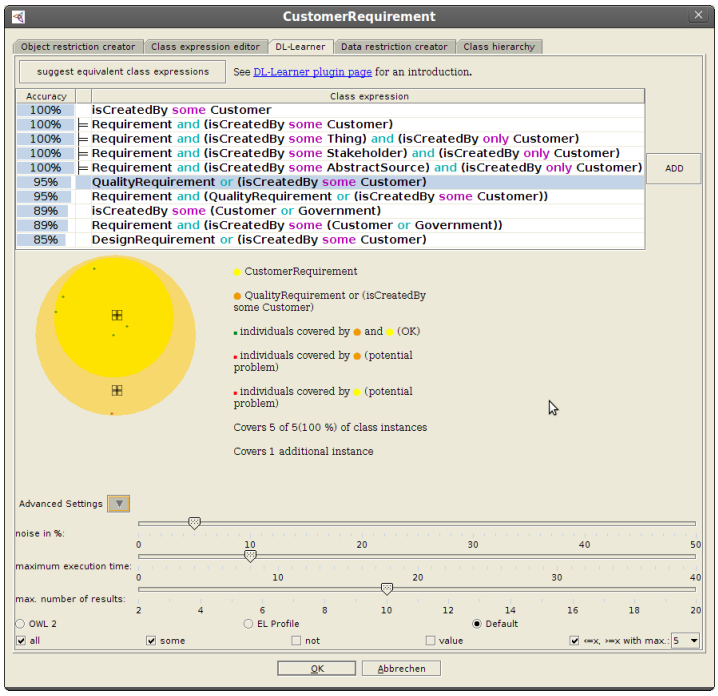


Figure 4. A screenshot of the DL-Learner Protégé plugin. It is integrated as additional tab to create class expressions in Protégé. The user is only required to press the "suggest equivalent class expressions" button and within a few seconds they will be displayed ordered by accuracy. If desired, the knowledge engineer can visualize the instances of the expression to detect potential problems. At the bottom, optional expert configuration settings can be adopted.

pletely, but in practice this will often not be the case. Clicking on the plus symbol in each circle shows its list of individuals. Those individuals are also presented as points in the circles and moving the mouse over such a point shows information about the respective individual. Red points show potential problems, where it is important to note that we use a closed world assumption to detect those. For instance, in our initial example in Section 1, a capital which is not related via the property `isCapitalOf` to an instance of `Country` is marked red. If there is not only a potential problem, but adding the expression would render the ontology inconsistent, the suggestion is marked red and a warning message is displayed. Accepting such a suggestion can still be a good choice, because the problem often lies elsewhere in the knowledge base, but was not obvious before, since the ontology was not sufficiently expressive for reasoners to detect it. This is illustrated by a screencast available from the plugin homepage, [6] where the ontology becomes inconsistent after adding the axiom, and the real source of the problem is fixed afterwards. Being able to make such suggestions can be seen as a strength of the plugin.

The plugin allows the knowledge engineer to change expert settings. Those settings include the maximum suggestion search time, the number of results returned and settings related to the desired target language., e.g. the knowledge engineer can choose to stay within the OWL 2 EL profile or enable/disable certain class expression constructors. The learning algorithm is designed to be able to handle noisy data and the visualisation of the suggestions will reveal false class assignments so that they can be fixed afterwards.

[6] `http://dl-learner.org/wiki/ProtegePlugin`

7 The OntoWiki Plugin

Analogous to Protégé, we created a similar plugin for OntoWiki [3]. OntoWiki is a lightweight ontology editor, which allows distributed and collaborative editing of knowledge bases. It focuses on wiki-like, simple and intuitive authoring of semantic content, e.g. through inline editing of RDF content, and provides different views on instance data.

Recently, a fine-grained plugin mechanism and extensions architecture was added to OntoWiki. The DL-Learner plugin is technically realised by implementing an OntoWiki component, which contains the core functionality, and a module, which implements the UI embedding. The DL-Learner plugin can be invoked from several places in OntoWiki, for instance through the context menu of classes as shown in Figure 5.

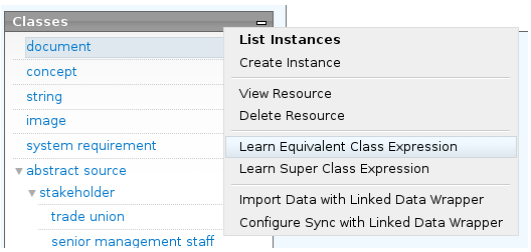


Figure 5. The DL-Learner plugin can be invoked from the context menu of a class in OntoWiki.

The plugin accesses DL-Learner functionality through its WSDL-based web service interface. Jar files containing all necessary libraries are provided by the plugin. If a user invokes the plugin, it scans whether the web service is online at its default address. If not, it is started automatically.

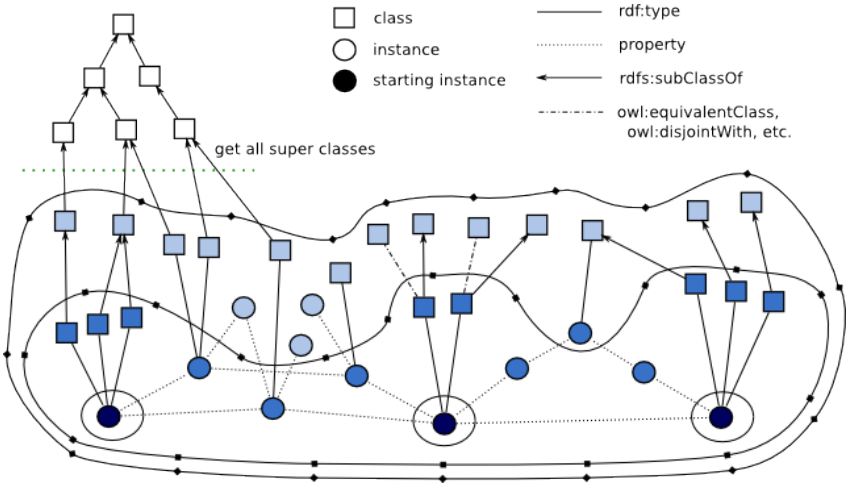


Figure 6. Extraction with three starting instances. The circles represent different recursion depths. The circles around the starting instances signify recursion depth 0. The larger inner circle represents the fragment with recursion depth 1 and the largest outer circle with recursion depth 2. Figure taken from [16].

A major technical difference compared to the Protégé plugin is that the knowledge base is accessed via SPARQL, since OntoWiki is a SPARQL-based web application. In Protégé, the current state of the knowledge base is stored in memory in a Java object. As a result, we cannot easily apply a reasoner on an OntoWiki knowledge base. To overcome this problem, we use the DL-Learner fragment selection mechanism described in [16]. Starting from a set of instances, the mechanism extracts a relevant fragment from the underlying knowledge base up to some specified recursion depth. Figure 6 provides an overview of the fragment selection process. The fragment has the property that learning results on it are similar to those on the complete knowledge base. For a detailed description we refer the reader to the full article.

The fragment selection is only performed for medium to large-sized knowledge bases. Small knowledge bases are retrieved completely and loaded into the reasoner. While the fragment selection can cause a delay of several seconds before the learning algorithm starts, it also offers flexibility and scalability. For instance, we can learn class expressions in large knowledge bases such as DBpedia in OntoWiki. [7]

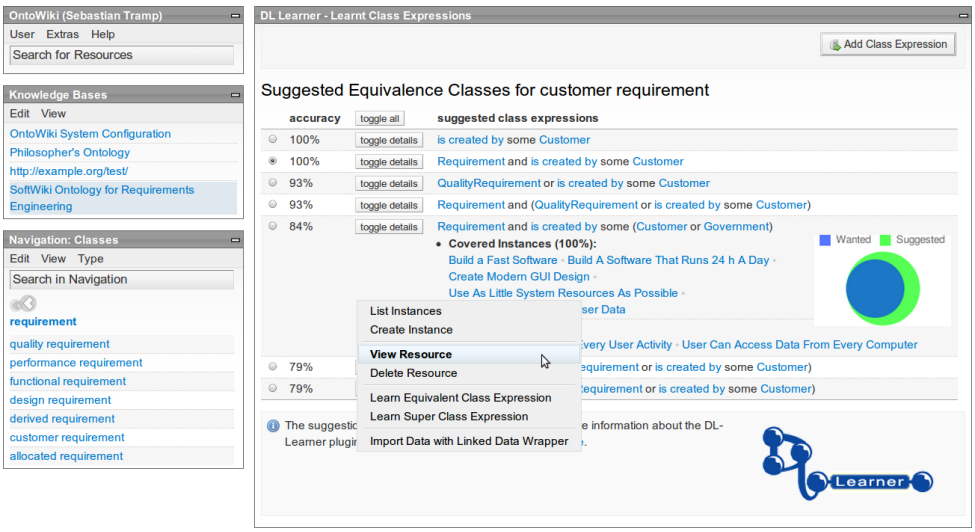


Figure 7. Screenshot of the result table of the DL-Learner plugin in OntoWiki.

Figure 7 shows a screenshot of the OntoWiki plugin applied to the SWORE [30] ontology. Suggestions for learning the class "customer requirement" are shown in Manchester OWL Syntax. Similar to the Protégé plugin, the user is presented a table of suggestions along with their accuracy value. Additional details about the instances of "customer requirement", covered by a suggested class expressions and additionally contained instances can be viewed via a toggle button. The modular design of OntoWiki allows rich user interaction: Each

[7] OntoWiki is undergoing an extensive development, aiming to support handling such large knowledge bases. A release supporting this is expected for the first half of 2012.

resource, e.g. a class, property, or individual, can be viewed and subsequently modified directly from the result table as shown for "design requirement" in the screenshot. For instance, a knowledge engineer could decide to import additional information available as Linked Data and run the CELOE algorithm again to see whether different suggestions are provided with additional background knowledge.

8 Evaluation

To evaluate the suggestions made by our learning algorithm, we tested it on a variety of real world ontologies of different sizes and domains. Please note that we intentionally do not perform an evaluation of the machine learning technique as such on existing benchmarks, since we build on the base algorithm already evaluated in detail in [26]. It was shown that this algorithm is superior to other supervised learning algorithms for OWL and at least competitive with the state of the art in ILP. Instead, we focus on its use within the ontology engineering scenario. The goals of the evaluation are to 1. determine the influence of reasoning and heuristics on suggestions, 2. to evaluate whether the method is sufficiently efficient to work on large real world ontologies, and 3. assess the accuracy of the approximation method presented in Section 4 in practice.

To perform the evaluation, we wrote a dedicated plugin for the Protégé ontology editor. This allows the evaluators to browse the ontology while deciding whether the suggestions made are reasonable. The plugin works as follows: First, all classes with at least 5 inferred instances are determined. For each such class, we run CELOE with different settings to generate suggestions for definitions. Specifically, we tested two reasoners and five different heuristics. The two reasoners are standard Pellet and Pellet combined with the instance check procedure described in Section 4. The five heuristics are those described in Section 3. For each configuration of CELOE, we generate at most 10 suggestions exceeding a heuristic threshold of 90%. Overall, this means that there can be at most 2 * 5 * 10 = 100 suggestions per class – usually less, because different settings of CELOE will still result in similar suggestions. This list is shuffled and presented to the evaluators. For each suggestion, the evaluators can choose between 6 options (see Table 3): 1.) the suggestion improves the ontology (improvement) 2.) the suggestion is no improvement and should not be included (not acceptable) and 3.) adding the suggestion would be a modelling error (error). In the case of existing definitions for class A, we removed them prior to learning. In this case, the evaluator could choose between three further options 4.) the learned definition is equal to the previous one and both are good (equal +), 5.) the learned definition is equal to the previous one and both are bad (equal -) and 6.) the learned definition is inferior to the previous

Ontology	#logical axioms	#classes	#object properties	#data properties	#individuals	DL expressivity
SC Ontology [8]	20081	28	8	5	3542	$\mathcal{AL}(\mathcal{D})$
Adhesome [9]	12043	40	33	37	2032	$\mathcal{ALCHN}(\mathcal{D})$
GeoSkills [10]	14966	613	23	21	2620	$\mathcal{ALCHOIN}(\mathcal{D})$
Eukariotic [11]	38	11	1	0	11	$\mathcal{ALCON}$
Breast Cancer [12]	878	196	22	3	113	$\mathcal{ALCROF}(\mathcal{D})$
Economy [13]	1625	339	45	8	482	$\mathcal{ALCH}(\mathcal{D})$
Resist [14]	239	349	134	38	75	$\mathcal{ALUF}(\mathcal{D})$
Finance [15]	16014	323	247	74	2466	$\mathcal{ALCROIQ}(\mathcal{D})$
Earthrealm [16]	931	2364	215	36	171	$\mathcal{ALCHO}(\mathcal{D})$

Table 2
Statistics about test ontologies

one (inferior).

We used the default settings of CELOE, e.g. a maximum execution time of 10 seconds for the algorithm. The knowledge engineers were five experienced members of our research group, who made themselves familiar with the domain of the test ontologies. Each researcher worked independently and had to make 998 decisions for 92 classes between one of the options. The time required to make those decisions was approximately 40 working hours per researcher. The raw agreement value of all evaluators is 0.535 (see e.g. [1] for details) with 4 out of 5 evaluators in strong pairwise agreement (90%). The evaluation machine was a notebook with a 2 GHz CPU and 3 GB RAM.

Table 3 shows the evaluation results. All ontologies were taken from the Protégé OWL [17] and TONES [18] repositories. We randomly selected 5 ontologies comprising instance data from these two repositories, specifically the

[8] http://www.mindswap.org/ontologies/SC.owl
[9] http://www.sbcny.org/datasets/adhesome.owl
[10] http://i2geo.net/ontologies/current/GeoSkills.owl
[11] http://www.co-ode.org/ontologies/eukariotic/2005/06/01/eukariotic.owl
[12] http://acl.icnet.uk/%7Emw/MDM0.73.owl
[13] http://reliant.teknowledge.com/DAML/Economy.owl
[14] http://www.ecs.soton.ac.uk/~aoj04r/resist.owl
[15] http://www.fadyart.com/Finance.owl
[16] http://sweet.jpl.nasa.gov/1.1/earthrealm.owl
[17] http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library
[18] http://owl.cs.manchester.ac.uk/repository/

reasoner/heuristic	improvement	equal quality (+)	equal quality (-)	inferior	not acceptable	error	missed improvements in %	selected position on suggestion list (incl. std. deviation)	avg. accuracy of selected suggestion in %
Pellet/F-Measure	16.70	0.44	0.66	0.00	64.66	17.54	14.95	2.82 ± 2.93	96.91
Pellet/Gen. F-Measure	15.24	0.44	0.66	0.11	66.60	16.95	16.30	2.78 ± 3.01	92.76
Pellet/A-Measure	16.70	0.44	0.66	0.00	64.66	17.54	14.95	2.84 ± 2.93	98.59
Pellet/pred. acc.	16.59	0.44	0.66	0.00	64.83	17.48	15.22	2.69 ± 2.82	98.05
Pellet/Jaccard	16.81	0.44	0.66	0.00	64.66	17.43	14.67	2.80 ± 2.91	95.26
Pellet FIC/F-Measure	36.30	0.55	0.55	0.11	52.62	9.87	1.90	2.25 ± 2.74	95.01
Pellet FIC/Gen. F-M.	33.41	0.44	0.66	0.00	53.41	12.09	7.07	1.77 ± 2.69	89.42
Pellet FIC/A-Measure	36.19	0.55	0.55	0.00	52.84	9.87	1.63	2.21 ± 2.71	98.65
Pellet FIC/pred. acc.	32.99	0.55	0.55	0.11	55.58	10.22	4.35	2.17 ± 2.55	98.92
Pellet FIC/Jaccard	36.30	0.55	0.55	0.11	52.62	9.87	1.90	2.25 ± 2.74	94.07

Table 3
Options chosen by evaluators aggregated by class.

Earthrealm, Finance, Resist, Economy and Breast Cancer ontologies (see Table 2).

Objective 1: The results in Table 3 show which options were selected by the evaluators. It clearly indicates that the usage of the Fast Instance Checker (FIC) in conjunction with Pellet (as described in Section 4) is sensible. The results are, however, more difficult to interpret with regard to the different employed heuristics. Using predictive accuracy did not yield good results and, surprisingly, generalised F-Measure also had a lower percentage of cases where option 1 was selected. The other three heuristics generated very similar results. One reason is that those heuristics are all based on precision and recall, but in addition the low quality of some of the randomly selected test ontologies posed a problem. In cases of too many very severe modelling errors, e.g. conjunctions and disjunctions mixed up in an ontology or inappropriate domain and range restrictions, the quality of suggestions decreases for each of the heuristics. This is the main reason why the results for the different heuristics are very close. Particularly, generalised F-Measure can show its strengths mainly for properly designed ontologies. For instance, column 2 of Table 3 shows that it missed 7% of possible improvements. This means that for 7% of all classes, one of the other four heuristics was able to find an appropriate definition, which was not suggested when employing generalised F-Measure. The last column in this table shows that the average value of generalised F-Measure is quite low. As explained previously, it distinguishes between cases when

an individual is instance of the observed class expression, its negation, or none of both. In many cases, the reasoner could not detect that an individual is instance of the negation of a class expression, because of the absence of disjointness axioms and negation in the knowledge base, which explains the low average values of generalised F-Measure. Column 4 of Table 3 shows that many selected expressions are amongst the top 5 (out of 10) in the suggestion list, i.e. providing 10 suggestions appears to be a reasonable choice.

In general, the improvement rate is only at about 35% according to Table 3 whereas it usually exceeded 50% in preliminary experiments with other real world ontologies with fewer or less severe modelling errors. Since CELOE is based on OWL reasoning, it is clear that schema modelling errors will have an impact on the quality of suggestions. As a consequence, we believe that the CELOE algorithm should be combined with ontology debugging techniques. We have obtained first positive results in this direction and plan to pursue it in future work. However, the evaluation also showed that CELOE does still work in ontologies, which probably were never verified by an OWL reasoner.

In the second part of our evaluation, we measured the performance of optimisation steps (see Section 4). In order to do so, we used a similar procedure as above, i.e. we processed the same ontologies and measured for how many class expressions we can measure a heuristic value within the execution time of 10 seconds. For each ontology, we averaged this over all classes with at least three instances. We did this for four different setups by enabling/disabling the stochastic heuristic measure and enabling/disabling the reasoner optimisations. We used Pellet 2.0 as underlying reasoner. The test machine had a 2.2 GHz dual core CPU and 4 GB RAM.

Objective 2: The results of our performance measurements are shown in Table 4. If evaluating a single class expression would take longer than a minute (the algorithm does not stop during such an evaluation), we did not add an entry to the table. We can observe that the approximate reasoner and the stochastic test procedure both lead to significant performance improvements. Since we prefer closed world reasoning as previously explained, the approximate reasoner would be a better choice even without improved performance. The performance gain of the stochastic methods is higher for larger ontologies. Apart from better performance on average, we also found that the time required to test a class expression shows smaller variations compared to the non-stochastic variant. Overall, a performance gain of several orders of magnitudes has been achieved. One can conclude that without approximate reasoning and stochastic coverage tests, the learning method would not work reasonably well on ontologies with large Aboxes.

Objective 3: To estimate the accuracy of the stochastic coverage tests, we evaluated each expression occurring in a suggestion list using the stochastic

and non-stochastic approach. The last column in Table 4 shows the average absolute value of the difference between these two values. It shows that the approximation differs by less than 1% on average from an exact computation of the score with low standard deviations. This means that the results we obtain through the method described in Section 4 are very accurate and there is hardly any influence on the learning algorithm apart from improved performance.

ontology	#tests stochastic		#tests non-stochastic		stoch. diff.
	appr. reas.	stand. reas.	appr. reas.	stand. reas.	(± std. dev.)
SC Ontology	21 000	—	670	—	0.5% ± 0.7%
Adhesome	22 000	38	1 850	—	0.2% ± 0.5%
Resist	67 000	6 600	30 000	1 500	0.0% ± 0.0%
Finance	67 000	72	27 000	—	0.2% ± 0.4%
GeoSkills	73 000	—	24 000	—	0.5% ± 0.7%
Earthrealm	79 000	2 200	39 500	490	0.0% ± 0.0%
Breast Cancer	113 000	2 700	81 000	1 200	0.2% ± 0.4%
Eukariotic	113 000	7 900	112 000	6 200	0.1% ± 0.1%
Economy	125 000	8 300	43 000	—	0.4% ± 0.6%

Table 4
Performance assessment showing the number of class expressions, which are evaluated heuristically within 10 seconds, with stochastic tests enabled/disabled and approximate reasoning enabled/disabled. The last column shows how much heuristic values computed stochastically differ from real values. Values are rounded.

9 Related Work

Related work can be categorised into two areas: The first one being supervised machine learning in OWL/DLs and the second being work on (semi-)automatic ontology engineering methods.

Early work on *supervised learning in description logics* goes back to e.g. [9,10], which used so-called *least common subsumers* to solve the learning problem (a modified variant of the problem defined in this article). Later, [7] invented a refinement operator for $\mathcal{ALER}$ and proposed to solve the problem by using a top-down approach. [14,19,20] combine both techniques and implement them in the YINYANG tool. However, those algorithms tend to produce very long and hard-to-understand class expressions. The algorithms implemented in DL-Learner [24,25,21,26] overcome this problem and investigate the learning problem and the use of top down refinement in detail. DL-FOIL [15] is a similar approach, which is based on a mixture of upward and downward refinement of class expressions. They use alternative measures in their evaluation. Instead of choosing different evaluation criteria, we decided to define a score

(see Section 3) to directly influence the search for solutions of the learning problem. Other approaches, e.g. [27] focus on learning in hybrid knowledge bases combining ontologies and rules. Ontology evolution [28] has been discussed in this context. Usually, hybrid approaches are a more general form of the work presented here, which enable learning powerful rules at the cost of a larger search space. The tradeoff between expressiveness of the target language and efficiency of learning algorithms is a critical choice in knowledge representation as well as in symbolic machine learning. In our work, which advances the research on applying ILP for ontology engineering towards its practical application, we decided to keep to the OWL 2 standard.

Regarding *(semi-)automatic ontology engineering*, the line of work starting in [31] and further pursued in e.g. [5] investigates the use of *formal concept analysis* for completing knowledge bases. It is promising, but targeted towards less expressive description logics and may not be able to handle noise as well as a machine learning technique. [33] proposes to improve knowledge bases through relational exploration and implemented it in the *RELExO framework* [19] . It is complementary to the work presented here, since it focuses on simple relationships and the knowledge engineer is asked a series of questions. The knowledge engineer either has to positively answer the question or provide a counterexample. A different approach to learning the definition of a named class is to compute the so called *most specific concept* (msc) for all instances of the class. The most specific concept of an individual is the most specific class expression, such that the individual is instance of the expression. One can then compute the *least common subsumer* (lcs) [6] of those expressions to obtain a description of the named class. However, in expressive description logics, an msc does not need to exist and the lcs is simply the disjunction of all expressions. For light-weight logics, such as $\mathcal{EL}$, the approach appears to be promising. [34] focuses on learning disjointness between classes in an ontology to allow for more powerful reasoning and consistency checking. In [13], inductive methods have been used to answer queries and populate ontologies using similarity measures and a *k-nearest neighbour* algorithm. Along this line of research, [11] defines similarity measures between concepts and individuals in description logic knowledge bases, which share similarities with the heuristic we defined in this article. Naturally, there is also a large amount of research work on ontology learning from text. The most closely related approach in this area is [32], in which OWL DL axioms are obtained by analysing sentences, which have definitional character.

[19] `http://code.google.com/p/relexo/`

10 Conclusions and Future Work

We presented the CELOE learning method specifically designed for extending OWL ontologies. Five heuristics were implemented and analysed in conjunction with CELOE along with several performance improvements. A method for approximating heuristic values has been introduced, which is useful beyond the ontology engineering scenario to solve the challenge of dealing with a large number of examples in ILP [35]. Furthermore, we biased the algorithm towards short solutions and implemented optimisations to increase readability of the suggestions made. The resulting algorithm was implemented in the open source DL-Learner framework. We argue that CELOE is the first ILP based algorithm, which turns the idea of learning class expressions for extending ontologies into practice. CELOE is integrated into two plugins for the ontology editors Protégé and OntoWiki and can be invoked using just a few mouse clicks.

We also performed a real world study on several ontologies to assess the performance of the algorithm in practice. While the results are generally promising, we also faced several hurdles due to significant modelling errors in the randomly selected ontologies. For this reason, we plan to investigate combinations of ontology debugging and class expression learning in future work. Preliminary experiments indicate that this can increase the percentage of improvements per class from about 35% to more than 50% with relatively simple modifications. Other areas of future work are the involvement of domain experts in the evaluation process, extensions towards the Web of Data, and combinations of RELExO and DL-Learner.

References

[1] A. Agresti, An Introduction to Categorical Data Analysis 2nd edition, Wiley-Interscience, 1997.

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

[3] S. Auer, S. Dietzold, T. Riechert, Ontowiki - a tool for social, semantic collaboration., in: ISWC 2006, vol. 4273 of LNCS, Springer, 2006.

[4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2007 (2007).

[5] F. Baader, B. Ganter, U. Sattler, B. Sertkaya, Completing description logic

knowledge bases using formal concept analysis, in: IJCAI 2007, AAAI Press, 2007.

[6] F. Baader, B. Sertkaya, A.-Y. Turhan, Computing the least common subsumer w.r.t. a background terminology, J. Applied Logic 5 (3) (2007) 392–420.

[7] L. Badea, S.-H. Nienhuys-Cheng, A refinement operator for description logics, in: ILP 2000, vol. 1866 of LNAI, Springer, 2000.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth, Occam's razor, in: Readings in Machine Learning, Morgan Kaufmann, 1990, pp. 201–204.

[9] W. W. Cohen, A. Borgida, H. Hirsh, Computing least common subsumers in description logics, in: AAAI 1992, 1992.

[10] W. W. Cohen, H. Hirsh, Learning the CLASSIC description logic: Theoretical and experimental results, in: KR 1994, Morgan Kaufmann, 1994.

[11] C. d'Amato, N. Fanizzi, F. Esposito, A semantic similarity measure for expressive description logics., in: CILC 2005, 2005.

[12] C. d'Amato, N. Fanizzi, F. Esposito, A note on the evaluation of inductive concept classification procedures, in: A. Gangemi, J. Keizer, V. Presutti, H. Stoermer (eds.), SWAP 2008, vol. 426 of CEUR Workshop Proceedings, CEUR-WS.org, 2008.

[13] C. d'Amato, N. Fanizzi, F. Esposito, Query answering and ontology population: An inductive approach, in: ESWC 2008, 2008.

[14] F. Esposito, N. Fanizzi, L. Iannone, I. Palmisano, G. Semeraro, Knowledge-intensive induction of terminologies from metadata, in: ISWC 2004, Springer, 2004.

[15] N. Fanizzi, C. d'Amato, F. Esposito, DL-FOIL concept learning in description logics, in: ILP 2008, vol. 5194 of LNCS, Springer, 2008.

[16] S. Hellmann, J. Lehmann, S. Auer, Learning of OWL class descriptions on very large knowledge bases, Int. J. Semantic Web Inf. Syst. 5 (2) (2009) 25–48.

[17] P. Hitzler, M. Krötzsch, S. Rudolph, Foundations of Semantic Web Technologies, Chapman & Hall/CRC, 2009.

[18] M. Horridge, P. F. Patel-Schneider, Manchester syntax for OWL 1.1, in: OWLED 2008, 2008.

[19] L. Iannone, I. Palmisano, An algorithm based on counterfactuals for concept learning in the semantic web, in: IEA/AIE 2005, 2005.

[20] L. Iannone, I. Palmisano, N. Fanizzi, An algorithm based on counterfactuals for concept learning in the semantic web, Applied Intelligence 26 (2) (2007) 139–159.

[21] J. Lehmann, Hybrid learning of ontology classes, in: Machine Learning and Data Mining in Pattern Recognition, vol. 4571 of LNCS, Springer, 2007.

[22] J. Lehmann, DL-Learner: learning concepts in description logics, Journal of Machine Learning Research (JMLR) 10 (2009) 2639–2642.

[23] J. Lehmann, Learning OWL class expressions, Ph.D. thesis, University of Leipzig, phD in Computer Science (2010).

[24] J. Lehmann, P. Hitzler, Foundations of refinement operators for description logics, in: ILP 2007, vol. 4894 of LNCS, Springer, 2008, best Student Paper Award.

[25] J. Lehmann, P. Hitzler, A refinement operator based learning algorithm for the ALC description logic, in: ILP 2007, vol. 4894 of LNCS, Springer, 2008, best Student Paper Award.

[26] J. Lehmann, P. Hitzler, Concept learning in description logics using refinement operators, Machine Learning journal 78 (1-2) (2010) 203–250.

[27] F. A. Lisi, Building rules on top of ontologies for the semantic web with inductive logic programming, Theory and Practice of Logic Programming 8 (3) (2008) 271–300.

[28] F. A. Lisi, F. Esposito, Learning SHIQ+log rules for ontology evolution, in: SWAP 2008, vol. 426 of CEUR Workshop Proceedings, CEUR-WS.org, 2008.

[29] S.-H. Nienhuys-Cheng, R. de Wolf (eds.), Foundations of Inductive Logic Programming, vol. 1228 of LNCS, Springer, 1997.

[30] T. Riechert, K. Lauenroth, J. Lehmann, S. Auer, Towards semantic based requirements engineering, in: I-KNOW 2007, 2007.

[31] S. Rudolph, Exploring relational structures via FLE, in: K. E. Wolff, H. D. Pfeiffer, H. S. Delugach (eds.), ICCS 2004, vol. 3127 of LNCS, Springer, 2004.

[32] J. Völker, P. Hitzler, P. Cimiano, Acquisition of OWL DL axioms from lexical resources, in: ESWC 2007, vol. 4519 of LNCS, Springer, 2007.

[33] J. Völker, S. Rudolph, Fostering web intelligence by semi-automatic OWL ontology refinement, in: Web Intelligence, IEEE, 2008.

[34] J. Völker, D. Vrandecic, Y. Sure, A. Hotho, Learning disjointness, in: ESWC 2007, vol. 4519 of LNCS, Springer, 2007.

[35] H. Watanabe, S. Muggleton, Can ILP be applied to large dataset?, in: ILP 2009, 2009.