Pattern Based Knowledge Base Enrichment

Lorenz Bühmann, Jens Lehmann

Universität Leipzig, Institut für Informatik, AKSW,
Postfach 100920, D-04009 Leipzig, Germany,
{buehmann|lehmann}@informatik.uni-leipzig.de
http://aksw.org

Abstract. Although an increasing number of RDF knowledge bases are published, many of those consist primarily of instance data and lack sophisticated schemata. Having such schemata allows more powerful querying, consistency checking and debugging as well as improved inference. One of the reasons why schemata are still rare is the effort required to create them. In this article, we propose a semi-automatic schemata construction approach addressing this problem: First, the frequency of axiom patterns in existing knowledge bases is discovered. Afterwards, those patterns are converted to SPARQL based pattern detection algorithms, which allow to enrich knowledge base schemata. We argue that we present the first scalable knowledge base enrichment approach based on real schema usage patterns. The approach is evaluated on a large set of knowledge bases with a quantitative and qualitative result analysis.

1 Introduction

Over the past years, the quantity and size of RDF knowledge bases has significantly increased. Nevertheless, many of those knowledge bases lack sophisticated schemata and instance data adhering to those schemata. For content extracted from legacy sources, crowdsourced content, but also manually curated content, it is challenging to ensure a co-evolution of schemata and data, in particular for large knowledge bases. For this reason, there has been significant recent interest in semi-automation of schemata creation and revision based on the available instance data [7,18,31,33]. The combination of instance data and schemata allows improved querying, inference and consistency checking. In particular, in previous work [7], we investigated lightweight and efficient schema creation approaches, which can scale to large knowledge bases. Furthermore, those methods are able to work with SPARQL based access to knowledge bases, which is currently the dominating form for querying knowledge bases, which cannot easily be handled by standard OWL reasoners. The main drawback of this early work is that we were limited to learn property axioms, e.g. domain and range. In this work, we go one step further and provide an approach, which is able to handle many frequent axiom types, while still being efficient. This is achieved by following a two phase approach: First, we analyse several data repositories to detect which terminological axiom patterns are frequently used and convert those patterns

to SPARQL queries, which help to find those axiom patterns in instance data. This preparation phase only needs to be performed once. Secondly, we perform a lightweight statistical analysis to actually find specific axiom candidates as suggestions for a knowledge engineer and compute a confidence score for each of them.

Example 1. As a running example for knowledge base enrichment, consider an axiom pattern[1]:

$$\mathtt{A} \equiv \mathtt{B} \sqcap \exists \mathtt{r}.\mathtt{C}$$

which can be instantiated by an axiom

$$\mathtt{SoccerPlayer} \equiv \mathtt{Person} \sqcap \exists \mathtt{team}.\mathtt{SoccerClub}$$

describing that every person which is in a team that is a soccer club, is a soccer player. Adding such an axiom to a knowledge base can have several benefits: 1.) The axioms serve as documentation for the purpose and correct usage of schema elements. 2.) They improve the application of constraint violation techniques. For instance, when using a tool such as the Pellet Constraint Validator[2] on a knowledge base with the above axiom, it would report soccer players without an associated team as violation.[3] 3.) Additional implicit information can be inferred, e.g. in the above example each person, who is in a soccer club team can be inferred to belong to the class `SoccerPlayer`, which means that an explicit assignment to that class is no longer necessary. The main purpose of our research is, therefore, to reduce the effort of creating and maintaining such schema information by providing enrichment suggestions to knowledge base maintainers.

We implemented our enrichment methods in the DL-Learner[4] framework [17] based on earlier work in [22,20]. The ORE tool [19][5] provides a graphical interface for them. Our main contributions are as follows:

- An analysis of 1392 ontologies containing approximately 20.5 million terminological axioms with respect to axiom patterns.
- Scalable retrieval and evaluation methods via SPARQL using sampling and confidence estimation.
- A manual evaluation of 11 patterns and 718 axioms in DBpedia [26].
- An open source implementation in DL-Learner.

The article is structured as follows: First, we present the overall workflow in Section 2. After that, the axiom normalisation into patterns and the estimation of their usage frequency is described in Section 3. Using [8] for converting such a

[1] We use standard description logic syntax in this paper and refer to [2] for an introduction.

[2] `http://clarkparsia.com/pellet/icv/`

[3] Under OWL semantics, this is not a violation, due to the Open World Assumption, unless we can infer from other knowledge that the player has no team.

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

[5] `http://ore-tool.net`

pattern to a SPARQL query, we show how to suggest new candidates for axioms, which could be added to the knowledge base in Section 4. For each suggestion, we provide a confidence value based on F-Score, which is detailed in Section 5. A performance optimisation for the workflow is outlined in Section 6. In Sections 7 and 8, we present our experimental setup and evaluation results, in particular we discuss benefits as well as cases in which our approach fails to provide correct suggestions. Related work is presented in Section 9 and we conclude in Section 10.

2 Knowledge Base Enrichment Workflow

In this section, we describe the overall workflow illustrated by Figure 1.

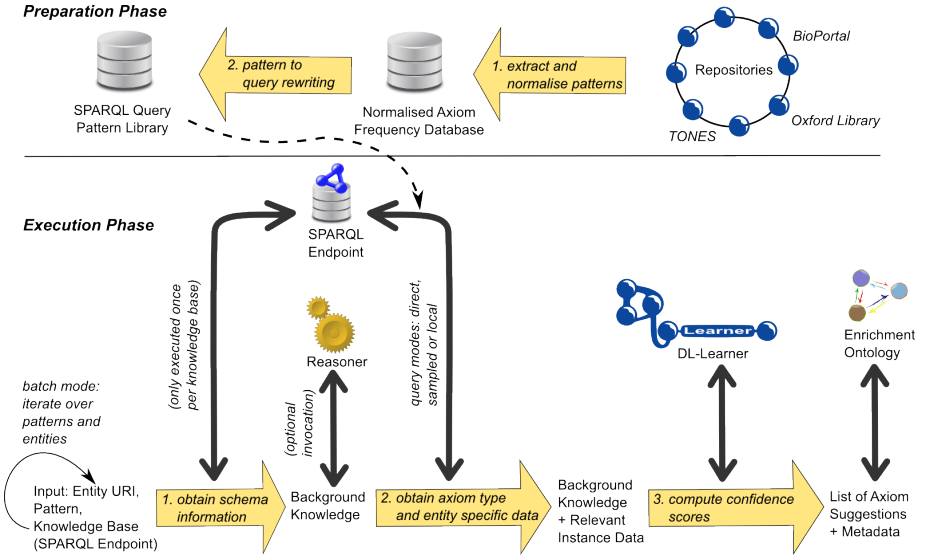

Fig. 1: Enrichment Workflow: In the preparation phase, typical axiom patterns are detected and converted to SPARQL queries, which are then used in the execution phase to learn new axiom suggestions.

The *preparation phase* (upper part), described in the next section, results in an automatically compiled library of query patterns for learning frequent axioms. Herein, the frequency is determined by analysing several ontology repositories and, afterwards, applying the method in [8] for converting the patterns to SPARQL queries.

In the *execution phase*, which is an extension of previous work [7], the actual axiom suggestions are generated. To achieve this, a single algorithm run takes an axiom pattern as input and generates a set of OWL axioms as a result. It proceeds in three steps:

1. In the optional first step, SPARQL queries are used to obtain existing information about the schema of the knowledge base, in particular we retrieve

axioms which allow to construct the class hierarchy. It can be configured whether to use an OWL reasoner for inferencing over the schema or just taking explicit knowledge into account.[6] Naturally, the schema only needs to be obtained once per knowledge base and can then be re-used by all algorithms and all entities.
2. The second step consists of obtaining data via SPARQL, which is relevant for learning the considered axiom. This results in a set of axiom candidates, configured via a threshold.
3. In the third step, the score of axiom candidates is computed and the results returned.

We will explain the preparation phase and steps 2 and 3 of the execution phase in the following sections in more detail by referring to our running example.

3 Pattern Frequency Detection

For detecting patterns, a set of input OWL files is used. Each axiom in those files is then transformed to a normal form, which we call an *axiom pattern*, which is defined as structural equivalence class[7].

An axiom is transformed to a pattern as follows: Let $Sig(ax)$ be the signature of an OWL axiom ax. Let C be an ordered list of named classes, P be an ordered list of properties and I be an ordered list of individuals. This order is then extended to class expressions using an ordering over the different types of class expressions. Based on this ordering, we can re-order the elements of intersection and disjunction expressions in axioms. After that, each element of $Sig(ax)$ is replaced by a placeholder variable.

As an example, this normalisation ensures that the axiom pattern $A \sqsubseteq B \sqcap \exists r.(C)$ is equally obtained from both $Father \sqsubseteq Person \sqcap \exists hasChild.Male$ and $Carnivore \sqsubseteq \exists eat.Meat \sqcap Animal$.

Naturally, there is no unique way to define patterns. For instance, in the above approach, we focus on patterns containing a single axiom. It would also be possible to detect sets of axioms, which combined result in typical usage patterns. At this stage, we did not do this due to scalability reasons: The algorithm needs to be able to read hundreds of potentially large files and, in a later stage, the generated SPARQL queries need to run on knowledge bases with billions of facts.

4 Data Retrieval

Usually, we have to run 3 SPARQL queries to obtain all data for computing the score by means of precision and recall: one query for the number of instances

[6] Note that the OWL reasoner only loads the schema of the knowledge base and, therefore, this option worked even in cases with several hundred thousand classes in our experiments using the HermiT reasoner.

[7] `http://www.w3.org/TR/owl2-syntax/#Structural_Specification`

of the left hand side ($|A|$), one for the instance count of the right hand side ($|B \sqcap \exists\ p.C|$), and another one to get the number of instances contained in the intersection of both ($|A \sqcap B \sqcap \exists\ p.C|$). Based on that information, we can compute precision P as $P = \frac{|A \sqcap B \sqcap \exists\, p.C|}{|B \sqcap \exists\, p.C|}$ and recall R as $R = \frac{|A \sqcap B \sqcap \exists\, p.C|}{|A|}$ in our example.

The transformation function defined in [8] applied to $A \sqcap B \sqcap \exists\ p.C$ (the intersection) leads to the following SPARQL query pattern:

```
?x a <A> .
?x a <B> .
?x <r> ?s0 .
?s0 a <C> .
```

Once having converted an axiom pattern into a SPARQL query pattern, in a next step we need to replace entities of the query pattern with variables V, resulting in another query pattern. Usually, the left hand side of the pattern represents the named classes in our knowledge base. For this reason, we can also iterate over all classes. This is not formally necessary, but in practice it splits up a large problem into subproblems, each of which requires less expensive SPARQL queries. Assuming that D is the current class, we obtain the following query:

```
?x a <D> .
?x a ?cls1 .
?x ?p ?s0 .
?s0 a ?cls2.
```

After that, we group and project the results to all entities which were replaced by variables in the previous step (`?p`, `?cls0`, `?cls1` in this case), and count the frequency for each combination. This results in a set of pattern instantiations and frequency counts. In Example 1, the corresponding is:

```
SELECT ?p ?cls0 ?cls1 (COUNT(DISTINCT(?x) as ?cnt)) WHERE
{ ?x a <D>.
  ?x a ?cls0.
  ?x ?p ?s0 .
  ?s0 a ?cls1
} GROUP BY ?p ?cls0 ?cls1 ORDER BY DESC(?cnt)
```

We assume that we do this for all three required queries (left hand side, right hand side, intersection).

5 Pattern Scoring

In the third workflow phase, we need to compute the confidence score for axiom candidates, which involves computing the F-measure for each candidate. For Example 1, computing the F-measure means that we need to count the number of instances which belong to `SoccerPlayer`, the number of instances of `Person`$\sqcap$ $\exists$`team.SoccerClub`, as well as the number of instances belonging to both, i.e. `SoccerPlayer` $\sqcap$ `Person` $\sqcap$ $\exists$`team.SoccerClub`. As explained above, the latter

value is divided on the one hand by the first value to obtain the recall, and on the other hand by the second value to get the precision, both resulting in a total score using standard F-measure. For our running example, assume that the following facts about some famous soccer players are given:

SoccerPlayer(Wayne_Rooney)
SoccerPlayer(Lionel_Messi)
Person(Wayne_Rooney)
Person(Lionel_Messi)
Person(Cristiano_Ronaldo)
SoccerClub(FC_Barcelona)
SoccerClub(Manchester_United_F.C.)
SoccerClub(Real_Madrid_C.F.)
team(Wayne_Rooney, Manchester_United_F.C.)
team(Lionel_Messi, FC_Barcelona)
team(Cristiano_Ronaldo, Real_Madrid_C.F.)

In the above example, we would obtain a recall of 100% (2 out of 2) and a precision of 66,7% (2 out of 3), resulting in a total F1 score of 80% for the pattern instantiation $\texttt{SoccerPlayer} \equiv \texttt{Person} \sqcap \exists \texttt{team}.\texttt{SoccerClub}$ of Example 1.

A disadvantage of using this straightforward method of obtaining a score is that it does not take the *support* for an axiom in the knowledge base into account. Specifically, there would be no difference between having 100 out of 100 correct observations or 3 out of 3 correct observations when computing precision and recall.

For this reason, we do not just consider the count, but the average of the 95% confidence interval of the count. This confidence interval can be computed efficiently by using the improved Wald method defined in [1]. Assume we have m observations out of which s were successful, then the approximation of 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 [1]. In the above case, this would change the precision to 57.3% (previously 66,7%) and the recall to 64.5% (previously 100%), thus leading to a total score of 60.7%. This indicates that there is not much support for the axiom in the knowledge base. However, when larger amounts of data are available, the score would increase and ultimately converge to standard F-score.

6 Optimizations

The disadvantage of the retrieval method in Section 4 is that it performs a *remote count* putting high computational load on the SPARQL endpoint in case of complex axiom patterns and very large data sets. An alternative option is to compute a *relevant fragment* of the knowledge base in a first step and then use that fragment to generate axiom candidates. We generate the relevant fragment as follows: Since we always had named classes on the left hand side of an axiom pattern, we iterate over all classes in the knowledge base. For each class A and axiom pattern $\mathfrak{p}$, we retrieve Concise Bounded Descriptions [8] of depth n for instances of A in a given time limit, where n is the modal depth of $\mathfrak{p}$. This is done via SPARQL CONSTRUCT queries and the result is loaded into a local triple store. We can then query this local store to obtain a local approximation of the recall value for specific instantiations of the axiom pattern. Each instantiation which is above a configurable threshold can then be processed by using the remote count method described in Section 4. In summary, this method performs a local filtering of axiom suggestions and only for viable candidates the exact precision and recall values are computed. The effect of this optimisation is that we reduce the query load on the triple store, in particular several simple queries are send instead of few very expensive queries, which could cause timeouts or overburden endpoints.

7 Experimental Setup

Pattern Frequency Detection There exists a number of well-known ontology repositories which are frequently used for empirical experimentation. For the pattern frequency detection step, we used the following repositories (details are listed in Table 1):

NCBO BioPortal[9]**,** an open repository of biomedical ontologies [28] that allows users to browse, search and visualize ontologies as well as to annotate and create mappings for ontologies. As of May 2013, the repository contains 385 ontologies in various ontology formats. Due to its ontologies ranging widely in size and complexity, the BioPortal has become a popular corpus for testing OWL ontology applications in recent years, such as pattern analysis [25] and ontology modularization [32].

TONES[10]**,** a curated ontology repository which was developed as part of the TONES project as a means of gathering suitable ontologies for testing OWL applications. It contains 219 well-known test and in-use ontologies, varying strongly in size (up to over 100,000 logical axioms) and complexity (from $\mathcal{EL}$++ to $\mathcal{SROIQ}$). The TONES ontologies are frequently used for empirical studies, such as the prediction of reasoning performance [15] and ontology debugging [19].

[8] CBD: `http://www.w3.org/Submission/CBD/`

Repository	#Ontologies		#Axioms							
	Total	Error	Total		Tbox		RBox		Abox	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
TONES	219	12	14,299	1,235,392	8297	658,449	20	932	5981	1,156,468
BioPortal	385	101	25,541	847,755	23,353	847,755	35	1339	2152	220,948
Oxford	793	0	49,997	2,492,761	15,384	2,259,770	25	1365	34,587	2,452,737

Table 1: Overview about the ontology repositories used in the experiments.

Oxford ontology library[11]**,** a collection of OWL ontologies which was, similar to the TONES repository, gathered for the purpose of testing OWL tools. The library which was established in late 2012 and currently contains 793 ontologies from 24 different sources, including an existing test corpus and several well-known in-use and test ontologies, the largest containing more than 2,000,000 axioms.

From the selected repositories, we used all ontologies which were available online and could be parsed by the OWL API[12], leading to 1392 ontologies containing approximately 20.5 million terminological axioms. From the ontologies that could not be processed (error column in Table 1), it was either not possible to load them from the given URL (TONES), they could not be parsed by the OWL API (TONES), or they were not publicly accessible (BioPortal).

Pattern Application For the evaluation of the pattern application, we used 100 randomly chosen classes with at least 5 instances of the well-known DBpedia (`http://dbpedia.org/sparql`) knowledge base, which is a crowd-sourced community effort to extract structured information from Wikipedia. In the used version (3.8), it contains facts about 3.77 million resources, many of them described by the 359 classes, 800 object properties and 859 datatype properties of the DBpedia ontology. We applied the optimization described in Section 6 with a time limit of 60 seconds for the fragment extraction process using thresholds of 0.6. From the results we showed at most 100 pattern instantiations per pattern to 3 non-author evaluators.

8 Results and Discussion

Pattern Frequency Detection As a result of the pattern frequency detection, we obtained an ordered list of the 15 most frequent non-trivial[13] TBox axiom patterns existing in at least 5 ontologies, as shown in Table 2. It shows how often each axiom pattern occurred (frequency), in how many ontologies it was contained, and the rank by frequency in each ontology repository. In addition, we also report the winsorised frequency: In the sorted list of pattern frequencies for each ontology (without 0-entries), we set all list entries higher than the 95th

[12] `http://owlapi.sourceforge.net/`

[13] $A \sqsubseteq \top$ and $A \sqsubseteq A$ were filtered

	Pattern	Frequency	Winsorized Frequency	#Ontologies	TONES	BioPortal	Oxford
	1. A $\sqsubseteq$ B	10,174,991	5,757,410	1050	2	1	1
	2. A $\sqsubseteq \exists$ p.B	8,199,457	2,450,582	604	1	2	2
	3. A $\sqsubseteq \exists$ p.($\exists$ q.B)	509,963	441,434	24	n/a	n/a	3
	4. A $\equiv$ B $\sqcap \exists$ p.C	361,777	316,420	319	8	4	4
*	5. B $\sqsubseteq \neg$ A	237,897	53,516	417	3	3	9
	6. A $\equiv$ B	104,508	8332	151	13	34	7
*	7. A $\equiv \exists$ p.B	70,040	11,031	139	36	32	8
	8. $\exists$ p.Thing $\sqsubseteq$ A	41,876	34,795	595	6	7	11
	9. A $\sqsubseteq \forall$ p.B	27,556	21,046	266	4	11	19
	10. A $\equiv$ B $\sqcap \exists$ p.C $\sqcap \exists$ q.D	24,277	20,277	196	11	13	13
	11. A $\equiv$ B $\sqcap$ C	16,597	16,597	78	5	20	22
	12. A $\sqsubseteq \exists$ p.(B $\sqcap \exists$ q.C)	12,453	12,161	84	23	18	15
	13. A $\sqsubseteq \exists$ p.{a}	11,816	4342	65	12	22	20
	14. A $\equiv$ B $\sqcap \exists$ p.(C $\sqcap \exists$ q.D)	10,430	10,430	60	39	21	17
*	15. p $\equiv q^-$	9943	7393	433	17	19	23

Table 2: Top 15 TBox axiom patterns ordered by frequency with additional information about the rank (if occurred) in each processed repository. Axiom patterns marked with * were omitted for the user evaluation.

percentile to the 95th percentile. This reduces the effect of outliers, i.e. axiom patterns scoring very high because of few very large ontologies frequently using them. The axioms marked with a star (*) are already covered by our previous work on learning an large knowledge bases [7]. We will use the remaining 12 patterns for our evaluation.

Fixpoint Analysis Based on the results of the pattern frequency detection, we performed a fixpoint analysis, i.e. we analysed how the ranking of the most frequent axiom patterns changed and, thus, investigated whether the ranking of the axiom patterns is fluctuating or stable. To do this, we processed ontology-by-ontology in random order and computed the current corresponding frequency ranking for each axiom pattern. The results shown in Figure 2 indicate that the ranking shows only minor changes after 300 ontologies and, hence, our input set of $\approx$ 1400 ontologies is sufficient.

Manual Evaluation Results Table 3 shows the result of the manual evaluation, which was done by 3 non-author evaluators. For the evaluation, we used a threshold score of 0.6. If more than 100 axioms were generated for a pattern type, we randomly selected 100 entries. This is shown as the sample size in Table 3. For pattern $A \sqsubseteq \forall p.B$, we could not find axioms above the threshold. Thus, we only evaluated the 11 remaining patterns. The last four columns are the result of a manual evaluation. All evaluators independently observed the sample manually and judged the axioms. An advantage of using DBpedia in this context is that

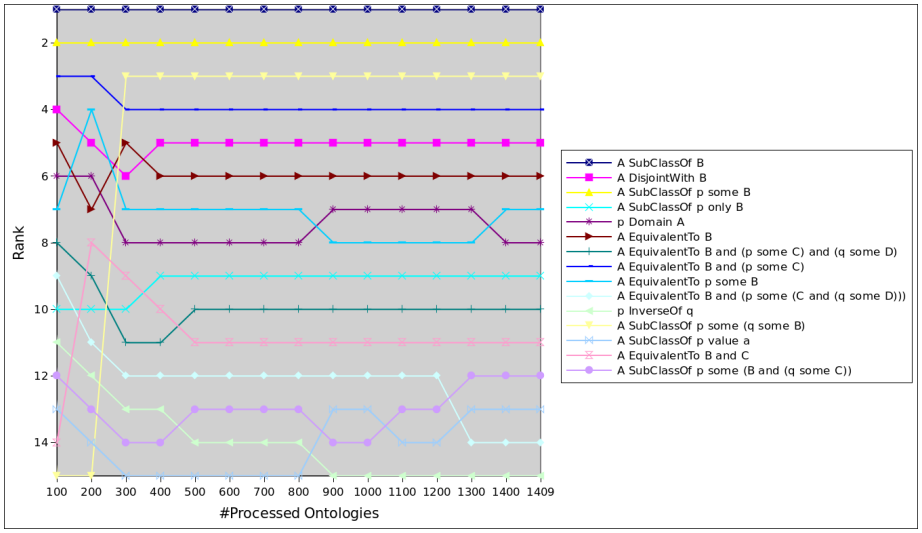

Fig. 2: Top 15 axiom patterns and its sequence of rank when processing the ontologies in random order.

the Wikipedia pages provide sufficient background knowledge in most cases in order to judge a particular axiom. Four different categories were used: "correct" indicates that it is likely that they would be accepted by a knowledge engineer, "minor" are axioms which are logically correct, but have modelling problems, "incorrect" are those, which contain conceptual flaws and "not judged" are axioms which could not be evaluated by the authors. Overall, out of 2154 decisions (718 evaluated axioms for each of the 3 reviewers), 48.2% were judged to be correct, 2.7% had minor issues, 49.0% were incorrect and 0 not judged. For a semi-automatic approach with manual validation, this is a reasonable score. The average interrater agreement was substantial, although it was poor for one axiom type. While half of the axiom patterns were frequent in DBpedia, one did not exist at all and 5 were infrequent.

Threshold Analysis The following diagram shows the correlation between the computed accuracy score of the pattern instantiations and the evaluator judgements, i.e. how many of the pattern instantiations with an accuracy value in a particular interval are correct using majority voting (at least 2 out of 3 reviewers have to judge it as correct).

To perform the analysis, questions were added in 10% buckets by confidence interval (60–70%, > 70% − 80%, > 80% − 90%, > 90% − 100%). Only buckets with at least 5 entries were used (which is why the lines are interrupted). For most axiom types, the trend is that axioms with higher confidence are more likely to be accepted, although two of the 11 axiom types show a decline with higher confidence. The overall trend (dashed line) shows a slope from approx. 50% to almost 60% indicating that higher accuracy scores result in better axiom suggestions.

pattern	sample size	manual evaluation in % correct	minor issues	incorrect	$\kappa_{\text{Fleiss'}}$
A $\sqsubseteq \exists$ p.B	50	88.0	0.7	11.3	24.8
A $\sqsubseteq$ B	47	63.8	2.1	34.0	53.8
A $\equiv$ B	25	10.7	0.0	89.3	44.0
A $\equiv \exists$ p.B	68	29.9	2.0	68.1	60.4
A $\equiv$ B $\sqcap \exists$ p.C	100	25.0	3.0	72.0	72.9
A $\equiv$ B $\sqcap \exists$ p.(C $\sqcap \exists$ q.D)	100	23.0	5.3	71.7	43.5
A $\sqsubseteq \exists$ p.($\exists$ q.B)	71	85.0	3.3	11.7	34.0
A $\sqsubseteq \exists$ p.(B $\sqcap \exists$ q.C)	100	87.0	0.3	12.7	-2.8
A $\sqsubseteq \exists$ p.{a}	15	71.1	0.0	28.9	45.9
A $\equiv$ B $\sqcap$ C	42	14.3	7.1	78.6	46.7
A $\equiv$ B $\sqcap \exists$ p.C $\sqcap \exists$ q.D	100	37.0	2.7	59.7	75.0
	718	48.2	2.7	49.0	66.1

Table 3: Result of the manual evaluation for each axiom pattern.

Discussion In this part, we will explain some of the evaluation results and present specific examples. In general, many axioms appeared to be close to human intuition. One of the reasons why axioms were often judged to have "minor issues" is that several suggestions for a particular class and axiom pattern were provided. The lower scoring ones often contained irrelevant parts. An example of this is `GrandPrix` $\equiv$ `Event` $\sqcap$ ($\exists$`poleDriver.Athlete`) $\sqcap$ ($\exists$`secondDriver.Agent`). While logically correct, defining a grand prix via the `Agent` class relation ship of the driver finishing second is not intuitive.

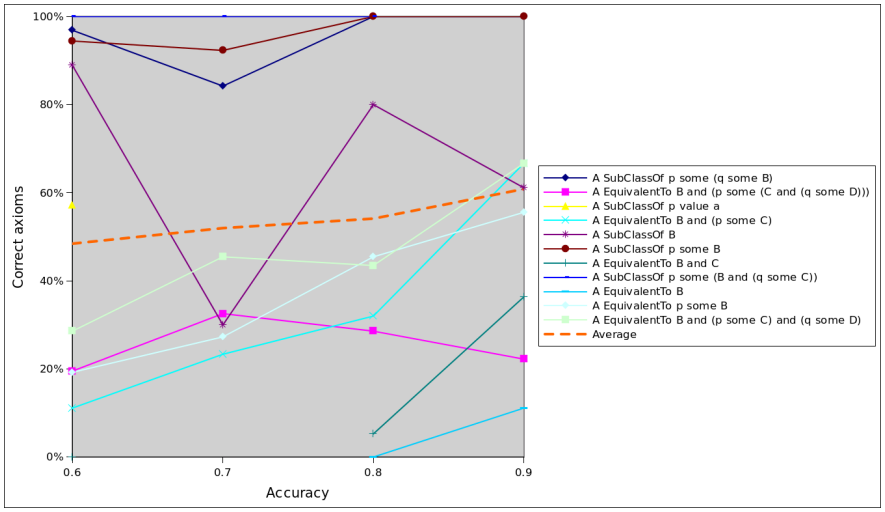

Fig. 3: Correlation between the accuracy value of the pattern instantiations and the confidence of the evaluators.

In other cases, there were conceptual flaws, e.g. in the following axioms:

1. $\texttt{Song} \sqsubseteq \exists \texttt{album}.\texttt{MusicalWork}$
2. $\texttt{Song} \sqsubseteq \exists \texttt{artist}.(\exists \texttt{recordLabel}.\texttt{RecordLabel})$
3. $\texttt{BritishRoyalty} \sqsubseteq \exists \texttt{parent}.\texttt{BritishRoyalty}$
4. $\texttt{President} \sqsubseteq \exists \texttt{successor}.\texttt{Person}$
5. $\texttt{SoccerManager} \equiv \texttt{Agent} \sqcap (\exists \texttt{birthPlace}.\texttt{Country}) \sqcap (\exists \texttt{managerClub}.\texttt{SportsTeam})$
6. $\texttt{SoccerClubSeason} \equiv \texttt{Organisation} \sqcap (\exists \texttt{manager}.\texttt{Person}) \sqcap (\exists \texttt{team}.\texttt{SoccerClub})$

The first axiom is not correct, because not each song actually appears on an album, although that is the case for the vast majority of songs in Wikipedia. Similarly, not each song is done by an artist having a record label. The third axiom is flawed, because to our understanding persons can marry British Royals and, e.g. become queen later on, without having been member of the royalty before. The fourth axiom is logically incorrect, because the current president does not have a successor yet. There are many successor relationships in DBpedia, which were suggested by our approach, so this was a major error type in the evaluation. The fifth axiom is also a typical example: The conceptual flaw is that a soccer manager has to manage a soccer team and not just an arbitrary sports team. This suggestion is generated, because soccer data is dominant in Wikipedia relative to other sports. However, in the best suggestion for the class `SoccerManager`, our approach provides the correct axiom. Finally, the last axiom is also incorrect. A soccer club season in DBpedia is modeled as a combination of a soccer club and a specific year, in which the team had a manager. However, `SoccerClubSeason` is a subclass of `Organisation`, which is a modeling error already in DBpedia itself. This particular modeling error had a negative influence on a significant number of axiom suggestions. Overall, the conceptual flaws are either axioms just above the threshold or those for which there is significant statistical evidence for their truth, but corner cases render them invalid. This is also the major reason why we believe that knowledge base construction cannot easily be fully automated.

For equivalent classes, there were 8 axioms above the 60% threshold. However, the DBpedia ontology does not contain classes, which could be seen as equivalent, so all axiom suggestions by our algorithm were classes suggested to be equivalent to their super classes due to having almost the same instances.

9 Related Work

Ontology Enrichment usually involves applying heuristics or machine learning techniques to find axioms, which can be added to an existing ontology. Naturally, different techniques have been applied depending on the specific type of axiom. One of the most complex tasks in ontology enrichment is to find *definitions* of classes. This is strongly related to Inductive Logic Programming (ILP) [27] and more specifically supervised learning in description logics. Early techniques [9] using *least common subsumers* were later enriched with refinement

operators [14]. However, those algorithms tend to produce very long and hard-to-understand class expressions. The algorithms implemented in DL-Learner [22] overcome this problem and investigate the learning problem and the use of top down refinement in detail. However, they require the ontology to be stored in an OWL reasoner in contrast to the work proposed in this article. DL-FOIL [10] is a similar approach, which is based on a mixture of upward and downward refinement of class expressions. Most recently, [18] implements appropriate heuristics and adaptations for learning definitions in ontologies.

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) [4] 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. Other approaches, e.g. [23] focus on learning in hybrid knowledge bases combining ontologies and *rules*.

Another enrichment task is *knowledge base completion*. The goal of such a task is to make the knowledge base complete in a particular well-defined sense. For instance, a goal could be to ensure that all subclass relationships between named classes can be inferred. The line of work starting in [29] and further pursued in e.g. [3] investigates the use of *formal concept analysis* for completing knowledge bases. [34] proposes to improve knowledge bases through relational exploration and implemented it in the *RELExO framework*[14]. It focuses on simple relationships and the knowledge engineer is asked a series of questions. The knowledge engineer either must positively answer the question or provide a counterexample.

[35] focuses on learning *disjointness* between classes in an ontology to allow for more powerful reasoning and consistency checking. To achieve this, it can use the ontology itself, but also texts, e.g. Wikipedia articles corresponding to a concept. The article includes an extensive study, which shows that proper modelling disjointness is actually a difficult task, which can be simplified via this ontology enrichment method.

There are further more light-weight ontology enrichment methods. For instance, *taxonomies* can be learned from simple tag structures via heuristics [7,33,31]. All of those approaches follow similar goals. [7] is the base of this article and follows the approach described in Section 2. [33] uses association rule mining with a different set of supported axioms. Learning in this settings is a batch process, which involves building transaction tables and measuring extensional overlap between classes. Finally, [31] follows similar idea, but is restricted to learning property domains and ranges as well as class disjointness. The approach is applied to inconsistency checking in DBpedia.

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

Type/Aim	References
Taxonomies	[36,7,33]
Definitions	often done via ILP approaches such as [21,22,18,10,5], genetic approaches [16] have also been used
Super Class Axioms	[18,33,7]
Rules in Ontologies	[23,24]
Disjointness	[35,7,31]
Properties of Properties	[7,11,31]
Completion	formal concept analysis and relational exploration [3,34,30]

Table 4: Work in ontology enrichment grouped by type or aim of learned structures.

Ontology Patterns There has been a significant amount of research on ontology design patterns with regular workshops on that topic. In particular, we want to refer to [13] for a systematic review on ontology design patterns articles in the Semantic Web community and [12] for a general introduction to the topic. Many patterns are listed at `http://ontologydesignpatterns.org`. Initially, we planned to use this as axiom pattern library. However, it turned out that only a small percentage of the patterns are applicable in the context of knowledge base enrichment and it is difficult to judge their relevancy. Therefore, we decided to perform the described bottom up approach involving several repositories and hundreds of ontologies. In the context of ontology learning, design patterns have been employed in [6]. However, the focus in that scenario is on developing ontologies from textual input, whereas our approach focuses on creating or refining schema structures from existing instance data.

10 Conclusions and Future Work

We presented an approach, which allows to detect frequent axiom usage patterns using $\approx$ 1400 ontologies and converted them into SPARQL query patterns allowing to find those patterns in instance data. This allows to improve knowledge base schemata semi-automatically and is the first scalable schema construction approach based on actual usage patterns to the best of our knowledge. Moreover, it improves the co-evolution of schema and data as well as querying, constraint checking and inference. The evaluation shows that the approach is feasible and able to provide useful suggestions. Nevertheless, we also pointed out corner cases, which are difficult to handle for such a statistical analysis and require human attention. In combination with previous efforts [7,18], we have build an efficient freely available tool, which is able to suggest both TBox and RBox axioms on large knowledge bases accessible via SPARQL endpoints.

Acknowledgement This work was supported by grants from the European Union's 7th Framework Programme provided for the projects GeoKnow (GA no. 318159) and LOD2 (GA no. 257943).

References

1. A. Agresti and B. A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.
2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
3. F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing description logic knowledge bases using formal concept analysis. In *IJCAI 2007*. AAAI Press, 2007.
4. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. Applied Logic*, 5(3):392–420, 2007.
5. L. Badea and S.-H. Nienhuys-Cheng. A refinement operator for description logics. In *ILP 2000*, volume 1866 of *LNAI*, pages 40–59. Springer, 2000.
6. E. Blomqvist. Ontocase-automatic ontology enrichment based on ontology design patterns. In *The Semantic Web-ISWC 2009*, pages 65–80. Springer, 2009.
7. L. Bühmann and J. Lehmann. Universal OWL axiom enrichment for large knowledge bases. In *Proceedings of EKAW 2012*, pages 57–71. Springer, 2012.
8. L. Bühmann and J. Lehmann. OWL class expression to SPARQL rewriting. Technical report, University of Leipzig, 2013. `http://svn.aksw.org/papers/2013/OWL_SPARQL/public.pdf`.
9. W. W. Cohen, A. Borgida, and H. Hirsh. Computing least common subsumers in description logics. In *AAAI 1992*, pages 754–760, 1992.
10. N. Fanizzi, C. d'Amato, and F. Esposito. DL-FOIL concept learning in description logics. In *ILP 2008*, volume 5194 of *LNCS*, pages 107–121. Springer, 2008.
11. D. Fleischhacker, J. Völker, and H. Stuckenschmidt. Mining rdf data for property axioms. In *OTM Conferences (2)*, pages 718–735, 2012.
12. A. Gangemi and V. Presutti. Ontology design patterns. In *Handbook on Ontologies*, pages 221–243. Springer, 2009.
13. K. Hammar and K. Sandkuhl. The state of ontology pattern research: a systematic review of iswc, eswc and aswc 2005–2009. In *Workshop on Ontology Patterns: Papers and Patterns from the ISWC workshop*, pages 5–17, 2010.
14. L. Iannone, I. Palmisano, and N. Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.
15. Y.-B. Kang, Y.-F. Li, and S. Krishnaswamy. Predicting reasoning performance using ontology metrics. In *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 198–214. Springer, 2012.
16. J. Lehmann. Hybrid learning of ontology classes. In *Proc. of the 5th Int. Conference on Machine Learning and Data Mining MLDM*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer, 2007.
17. J. Lehmann. DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642, 2009.
18. J. Lehmann, S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9:71 – 81, 2011.
19. J. Lehmann and L. Bühmann. ORE - a tool for repairing and enriching knowledge bases. In *Proceedings of the 9th International Semantic Web Conference (ISWC2010)*, Lecture Notes in Computer Science, pages 177–193. Springer, 2010.
20. J. Lehmann and P. Hitzler. Foundations of refinement operators for description logics. In H. Blockeel, J. Ramon, J. W. Shavlik, and P. Tadepalli, editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2007. Best Student Paper Award.

21. J. Lehmann and P. Hitzler. A refinement operator based learning algorithm for the $\mathcal{ALC}$ description logic. In *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2007. Best Student Paper Award.
22. J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250, 2010.
23. 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):271–300, 2008.
24. F. A. Lisi and F. Esposito. Learning SHIQ+log rules for ontology evolution. In *SWAP 2008*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
25. E. Mikroyannidi, N. A. A. Manaf, L. Iannone, and R. Stevens. Analysing syntactic regularities in ontologies. In P. Klinov and M. Horridge, editors, *OWLED*, volume 849 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.
26. M. Morsey, J. Lehmann, S. Auer, C. Stadler, and S. Hellmann. DBpedia and the Live Extraction of Structured Data from Wikipedia. *Program: electronic library and information systems*, 46:27, 2012.
27. S.-H. Nienhuys-Cheng and R. de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *LNCS*. Springer, 1997.
28. D. L. Rubin, D. A. Moreira, P. Kanjamala, and M. A. Musen. Bioportal: A web portal to biomedical ontologies. In *AAAI Spring Symposium: Symbiotic Relationships between Semantic Web and Knowledge Engineering*, pages 74–77. AAAI, 2008.
29. S. Rudolph. Exploring relational structures via FLE. In K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach, editors, *ICCS 2004*, volume 3127 of *LNCS*, pages 196–212. Springer, 2004.
30. B. Sertkaya. OntocomP system description. In B. C. Grau, I. Horrocks, B. Motik, and U. Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
31. G. Töpper, M. Knuth, and H. Sack. Dbpedia ontology enrichment for inconsistency detection. In *Proceedings of the 8th International Conference on Semantic Systems*, pages 33–40. ACM, 2012.
32. C. D. Vescovo, D. Gessler, P. Klinov, B. Parsia, U. Sattler, T. S. 0002, and A. Winget. Decomposition and modular structure of bioportal ontologies. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2011.
33. J. Völker and M. Niepert. Statistical schema induction. In G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, editors, *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2011.
34. J. Völker and S. Rudolph. Fostering web intelligence by semi-automatic OWL ontology refinement. In *Web Intelligence*, pages 454–460. IEEE, 2008.
35. J. Völker, D. Vrandecic, Y. Sure, and A. Hotho. Learning disjointness. In *ESWC 2007*, volume 4519 of *LNCS*, pages 175–189. Springer, 2007.
36. H. Wu, M. Zubair, and K. Maly. Harvesting social knowledge from folksonomies. In *Proceedings of the seventeenth conference on Hypertext and hypermedia*, HYPERTEXT '06, pages 111–114, New York, NY, USA, 2006. ACM.