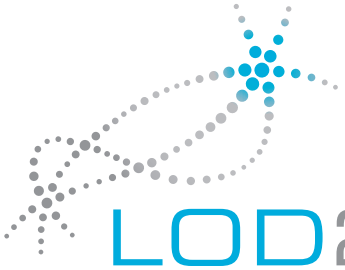
LOD2 Deliverable D3.3.3: Evaluation of Knowledge Base Enrichment

Lorenz Bühmann, Jens Lehmann

Abstract: Over the past years, the quantity and size of RDF knowledge bases has significantly increased - partially due to the success of the Linked Data initiative. However, many of those knowledge bases are not well structured. This deliverable describes learning algorithms for large knowledge bases which are accessible via SPARQL. The main focus of the report is the evaluation of those algorithms: Several hundred axioms were evaluated using independent evaluators on the DBpedia knowledge base.



Collaborative Project

LOD2 - Creating Knowledge out of Interlinked Data

Project Number: 257943 **Start Date of Project:** 01/09/2010 **Duration:** 48 months

Deliverable 3.3.3
Evaluation of Knowledge Base Enrichment

Dissemination Level	Public
Due Date of Deliverable	Month 36, 31/08/2013
Actual Submission Date	31/08/2012
Work Package	WP3, Evaluation of Knowledge Base Enrichment
Task	T 3.3
Type	Report
Approval Status	Final
Version	1.0
Number of Pages	38
Filename	deliverable-3.3.3.pdf

Abstract: This is a deliverable describing a general methodology for knowledge base enrichment algorithms, as well as their evaluation.

Project funded by the European Commission within the Seventh Framework Programme (2007 - 2013)

History

Version	Date	Reason	Revised by
0.0	10/07/2013	Initial Version	Lorenz Bühmann
0.5	20/07/2013	General Description of Enrichment	Lorenz Bühmann
0.9	15/08/2013	Peer review	Sebastian Hellmann
1.0	30/08/2013	Address peer review comments	Jens Lehmann

Author List

Organization	Name	Contact Information
ULEI	Lorenz Bühmann	buehmann@informatik.uni-leipzig.de
ULEI	Jens Lehmann	lehmann@informatik.uni-leipzig.de

Executive Summary

Over the past years, the quantity and size of RDF knowledge bases has significantly increased - partially due to the success of the Linked Data initiative. However, many of those knowledge bases are not well structured. This deliverable describes learning algorithms for large knowledge bases which are accessible via SPARQL. The main focus of the report is the evaluation of those algorithms: Several hundred axioms were evaluated using independent evaluators on the DBpedia knowledge base.

Table of Contents

List of Figures

1 Introduction and Motivation

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 [30, 47, 49]. The combination of instance data and schemata allows improved querying, inference and consistency checking.

For this deliverable, 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. We provide an approach, which is able to handle simple property axioms, as well as also frequently used complex 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. We use these patterns to automatically generate SPARQL queries, which help to find these 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 1.1 *As a first example, consider a knowledge base containing a property* ***birthPlace*** *and subjects in triples of this property, e.g. Brad Pitt, Angela Merkel, Albert Einstein etc. Enrichment algorithms could suggest that the property* ***birthPlace*** *may be functional and has the domain* ***Person*** *as it is encoded via the following axioms in Manchester OWL syntax*[1]*:*

```
ObjectProperty: birthPlace
    Characteristics: Functional
    Domain: Person
    Range: Place
    SubPropertyOf: hasBeenAt
```

Example 1.2 *As a second example for knowledge base enrichment, consider an axiom pattern*[2]*:*

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

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[3] *on a knowledge base with the above axiom, it would report soccer players without an associated team as violation.*[4] *3.) Additional implicit information*

[1] For details on Manchester OWL syntax (e.g. used in Protégé, OntoWiki) see http://www.w3.org/TR/owl2-manchester-syntax/.

[2] This example uses standard description logic syntax. We refer to [3] for an introduction.

[3] http://clarkparsia.com/pellet/icv/

[4] 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.

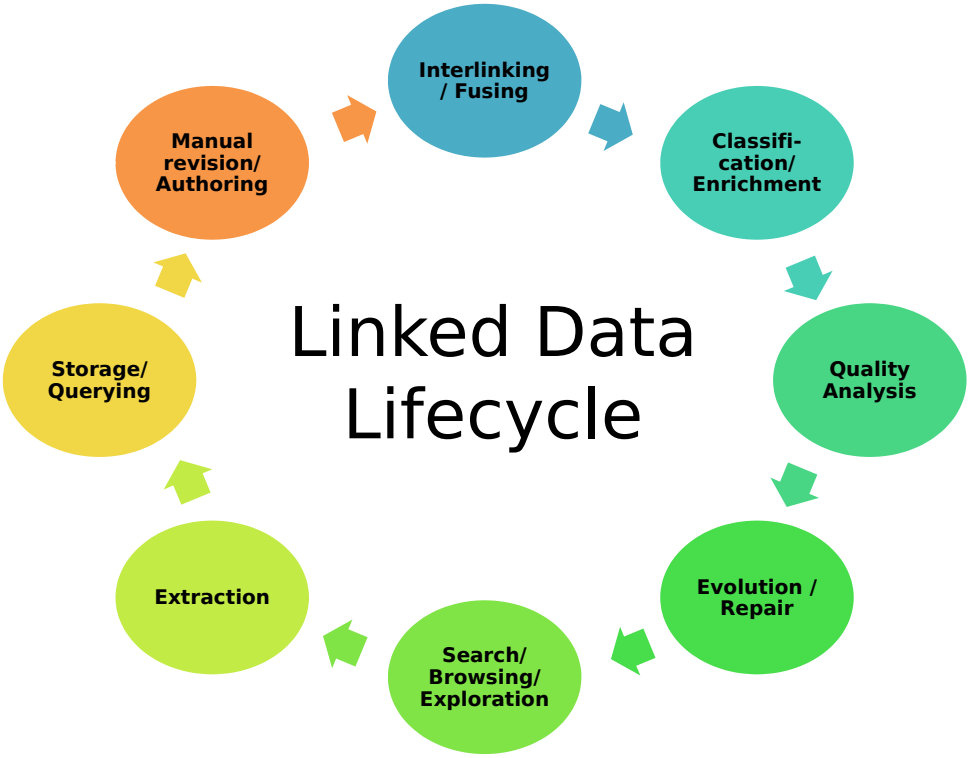


Figure 1: The approach presented in the deliverable covers the *enrichment phase* in the Linked Data Life-Cycle and is supported by the DL-Learner and ORE tools as part of the LOD2 Stack.

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[5] framework [28] based on earlier work in [37, 35]. The ORE tool [32][6] provides a graphical interface for them as described in the previous deliverable. Our main contributions are as follows:

- An analysis of 1392 ontologies containing approximately 20.5 million terminological axioms with respect to axiom patterns.
- An automatic conversion method from property axioms and more complex axiom types to SPARQL query patterns.
- Scalable retrieval and evaluation methods via SPARQL using sampling and confidence estimation.
- A manual evaluation using DBpedia [31] with 3 independent evaluators based on several hundred suggested axioms.
- An open source implementation in DL-Learner.

[5] http://dl-learner.org
[6] http://ore-tool.net

The deliverable 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 the method in Section 4 for converting such a pattern to a SPARQL query, we show how to suggest new candidates for axioms, which could be added to the knowledge base in Section 5. For each suggestion, we provide a confidence value based on F-Score, which is detailed in Section 6. A performance optimisation for the workflow is outlined in Section 7. In Sections 8 and 9, 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 10 and we conclude in Section 11.

2 Knowledge Base Enrichment Workflow

In this section, we describe the overall workflow illustrated by Figure 2.

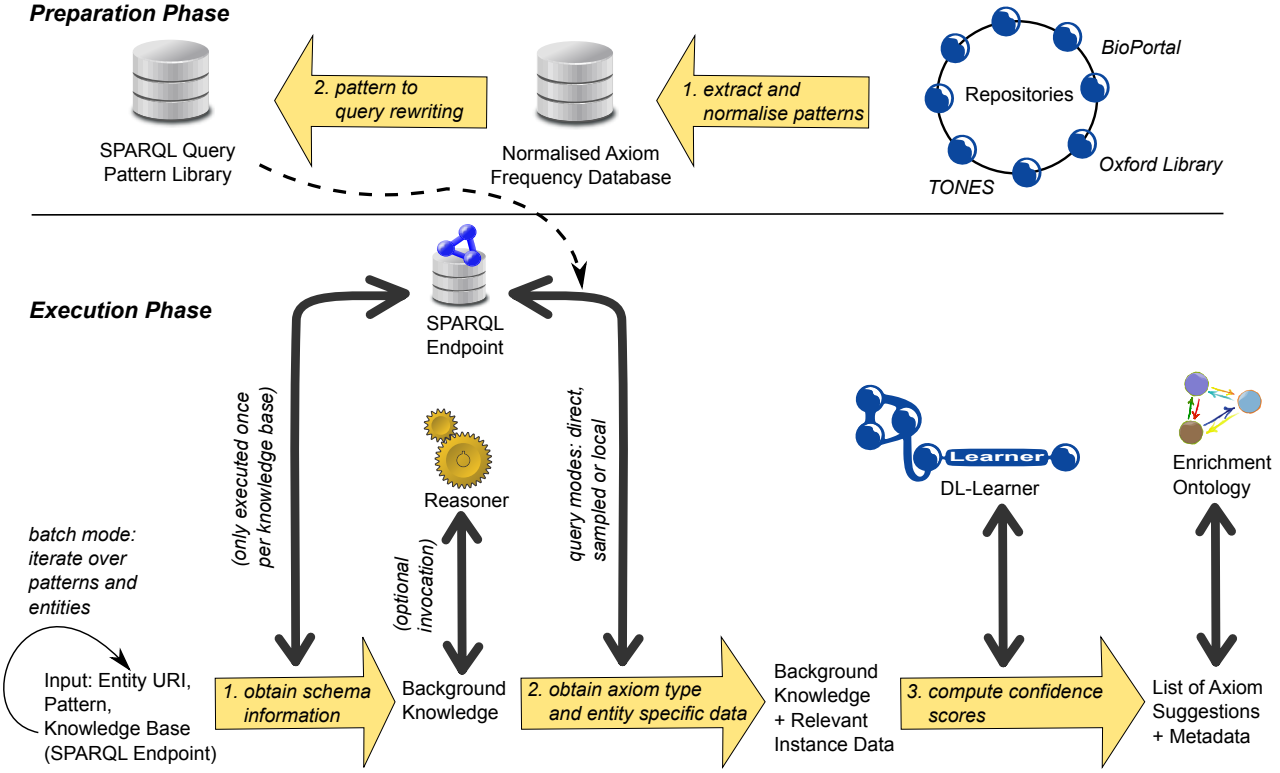


Figure 2: Enrichment Workflow: In the preparation phase, typical axiom patterns are detected and converted to SPARQL queries, which are then used in the execution phase to learn new axiom suggestions.

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 [9] for converting the patterns to SPARQL queries.

In the *execution phase*, which is an extension of previous work [8], 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.[7] 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.

[7]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.

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[8].

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.

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

4 Pattern Transformation

We restrict the description of the pattern rewriting algorithm for brevity to SubClassOf axiom patterns, as it was the most frequently used schema axiom type we discovered during the evaluation, and secondly each TBox axiom can be rewritten as a set of SubClassOf axioms.

Let $C, D, C_1, \ldots, C_n$ be class expressions, A be an atomic class, r be an object property, $a_1, \ldots, a_n$ be individuals and $\Theta = \{\leq, \geq, =\}$. Let $\tau(C, v)$ be a mapping from a class expression C and a target variable v to a SPARQL graph pattern $\mathfrak{p}$ as described in Table 1. A given axiom pattern $C \sqsubseteq D$ can be converted into a SPARQL query pattern $\mathfrak{p}$ by recursively applying τ on the class expression $C \sqcap D$, i.e. $\mathfrak{p} := \tau(C \sqcap D)$. We assume that τ generates unused query variables in its recursion.

The purpose of the SPARQL query is to determine whether instance data is structured according to the axiom pattern. Naturally, the SPARQL queries are only an approximation and do not employ the OWL open world assumption or use reasoning unless provided by the SPARQL endpoint[9]. The goal is not to infer axioms, but rather to detect statistical evidence for axioms in the ABox.

[9] See http://www.w3.org/TR/sparql11-entailment/.

Class Expression C_i	Graph Pattern $\mathfrak{p} = \tau(C_i,$?var$)$
A	{?var a A.}
$\neg C$	{?var ?p ?o . FILTER NOT EXISTS {$\tau(C,$?var)}}
$\{a_1, \ldots, a_n\}$	{?var ?p ?o . FILTER (?var IN ($a_1, \ldots, a_n$))}
$C_1 \sqcap \ldots \sqcap C_n$	{$\tau(C_1,$?var) $\cup \ldots \cup$ $\tau(C_n,$?var)}
$C_1 \sqcup \ldots \sqcup C_n$	{$\tau(C_1,$?var)} UNION ... UNION {$\tau(C_n,$?var)}
$\exists\, r.C$	{?var r ?s.} $\cup$ $\tau(C,$?s)
$\exists\, r.\{a\}$	{?var r a.}
$\forall\, r.C$	{ ?var r ?s0. { SELECT ?var (count(?s1) AS ?cnt1) WHERE { ?var r ?s1 . $\tau(C, ?s1)$ } GROUP BY ?var } { SELECT ?var (count(?s2) AS ?cnt2) WHERE { ?var r ?s2 } GROUP BY ?var } FILTER (?cnt1 = ?cnt2) }
$\Theta n\ r.C$	{ ?var r ?s0. { SELECT ?var WHERE { ?var r ?s . $\tau(C, ?s)$ } GROUP BY ?var HAVING (count(?s) Θ n) } }

Table 1: Conversion of class expressions into a SPARQL graph pattern.

5 Data Retrieval

5.1 Property Axioms

Given a property `$property`, we have to run two SPARQL queries to obtain all data for computing the score: one query for the number of triples for the given property (f_{total}), and another one returning the frequency of the particular axiom type (f_{axiom}), each of them described in the following subsections.

Based on that information, we can then compute the score by simply calculating $\frac{f_{\text{axiom}}}{f_{\text{total}}}$

5.1.1 Property Subsumption/Disjointness

For properties, learning subsumption and disjointness is analogous to learning this kind of axioms for classes. The difference is that we count how often subject `?s` and object `?o` in the triples for a given property `$property` are also related via other properties `?p`.

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

5.1.2 Property Domain and Range

For domains of object properties and data properties we count the occurrences of types in the subject position of triples having the property.

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

For property ranges, we issue different queries depending on whether a resource is a data or object property.

Object Properties

The object property case is analogous to learning domains as shown above, except this time we pay attention to the triple objects.

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

Data Properties

For data properties, we make use of the fact that every triple is annotated with its datatype in RDF, i.e. we can just count occurring datatypes.

```
SELECT ?datatype COUNT(DISTINCT ?ind) WHERE {
   ?ind <$property> ?val.
} GROUP BY (DATATYPE(?val) AS ?datatype)
```

5.1.3 Inverse Properties

To generate axioms which state that a property `p1` is the inverse of a property `p2` we run the query below, which retrieves properties `p` having subject and object occurring in the triples for the given property `$property` in swapped positions and count how often this happens.

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

5.1.4 Property Characteristics

Based on the axiom type which shall be learned for a given property `$property`, either the number of triples (symmetry, asymmetry), the number of distinct subjects (functionality, reflexivity, irreflexivity), the number of distinct objects (inverse-functionality) or the number of connected triple pairs (transitivity) is computed in a first step. This value is then combined with the result of the corresponding query in Table 2 .

5.2 Axiom Patterns

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 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 Section 4 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

Characteristic	SPARQL Query
Functionality	`SELECT COUNT(DISTINCT ?s) AS ?functional WHERE { ?s <$property> ?o1. FILTER NOT EXISTS {?s <$property> ?o2. FILTER(?o1 != ?o2)} }`
Inverse-Functionality	`SELECT COUNT(DISTINCT ?o) AS ?inversefunctional WHERE { ?s1 <$property> ?o. FILTER NOT EXISTS {?s2 <$property> ?o. FILTER(?s1 != ?s2)} }`
Symmetry	`SELECT (COUNT(*) AS ?symmetric) WHERE { ?s <$property> ?o. ?o <$property> ?s. }`
Asymmetry	`SELECT (COUNT(*) AS ?asymmetric) WHERE { ?s <$property> ?o. FILTER NOT EXISTS {?o <$property> ?s.} }`
Reflexivity	`SELECT (COUNT(DISTINCT ?s) AS ?reflexive) WHERE { ?s <$property> ?s. }`
Irreflexivity	`SELECT (COUNT(DISTINCT ?s) AS ?irreflexive) WHERE { ?s <$property> ?o. FILTER NOT EXISTS {?s <$property> ?s.} }`
Transitivity	`SELECT (COUNT(*) AS ?transitive) WHERE { ?s <$property> ?o. ?o <$property> ?o1. ?s <$property> ?o1. }`

Table 2: SPARQL queries used to learn the different types of OWL 2 property characteristics.

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.
```

corresponding query would be:

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 1.2, 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).

6 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 1.2, 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 `SoccerPlayer` $\equiv$ `Person`$\sqcap\exists$`team.SoccerClub` of Example 1.2.

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.

Note that in this implementation, more general classes in the hierarchy would always score higher. It might be desirable to correct this by slightly penalising very general classes. The drawback of such a penalty could be that the scores would be more difficult to understand for users. We leave the decision on such a penalty and a possible implementation as an area for future work.

7 Optimizations

The disadvantage of the retrieval method in Section 5 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 [10] 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 5. 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.

[10] 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 3: Overview about the ontology repositories used in the experiments.

8 Experimental Setup

8.1 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 3):

NCBO BioPortal[11]**,** an open repository of biomedical ontologies [43] 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 [40] and ontology modularization [48].

TONES[12]**,** 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 [26] and ontology debugging [32].

Oxford ontology library[13]**,** 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[14], leading to 1392 ontologies containing approximately 20.5 million terminological axioms. From the ontologies that could not be processed (error column in Table 3), 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).

[14] http://owlapi.sourceforge.net/

8.2 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`) [41, 2, 31] 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 7 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.

		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.	$\exists\ p.\text{Thing} \sqsubseteq A$	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 4: 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.

9 Results and Discussion

9.1 Pattern Frequency Detection

As a result of the pattern frequency detection, we obtained an ordered list of the 15 most frequent non-trivial[15] TBox axiom patterns existing in at least 5 ontologies, as shown in Table 4. 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 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 [8]. We will use the remaining 12 patterns for our evaluation.

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

9.2 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 3 and 4 indicate that the ranking shows only minor changes after 300 ontologies and, hence, our input set of ≈ 1400 ontologies is sufficient.

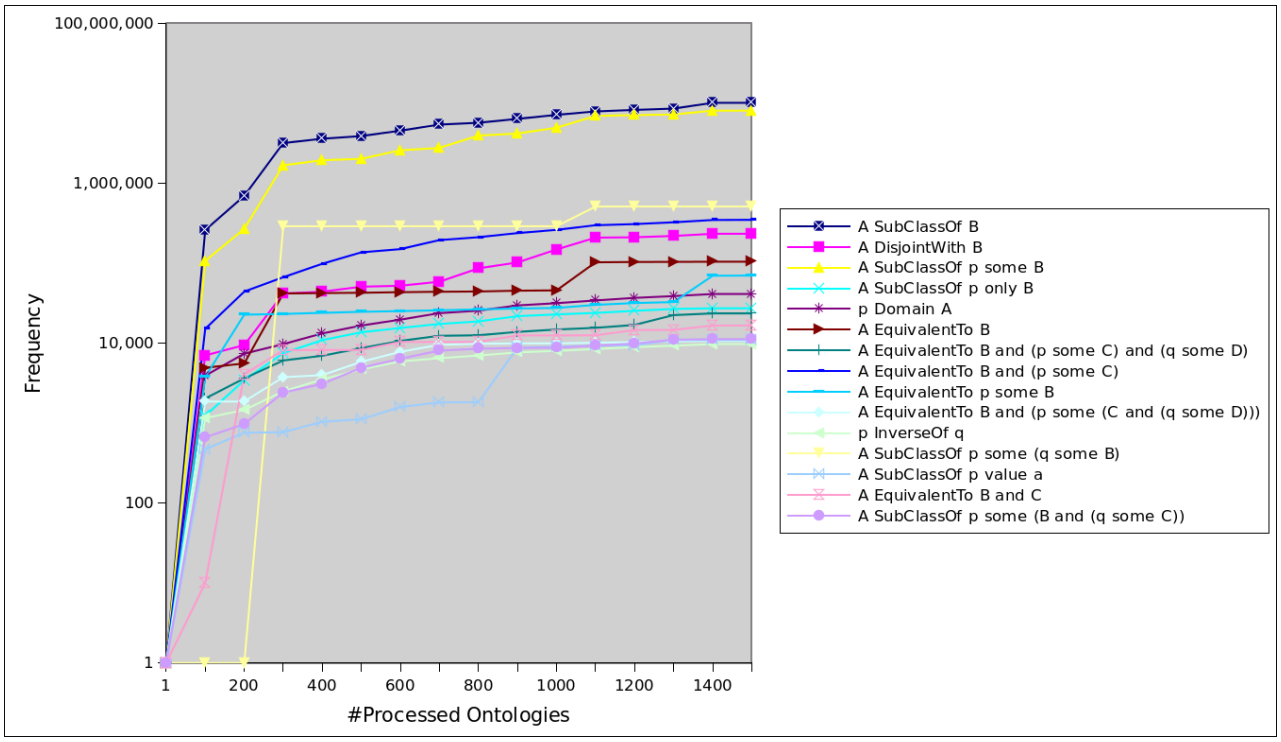


Figure 3: Top 15 axiom patterns and its sequence of total frequency when processing the ontologies in random order.

9.3 Manual Evaluation Results

9.3.1 Property Axioms

We performed an enrichment on the DBpedia Live knowledge base [41], which at that time consisted of 385 million triples, 3.64 million things, 272 classes, 629 object properties and 706 data properties. We used a confidence threshold of 0.7 for the algorithm runs and showed at most 10 suggestions per entity and axiom type. Table 5 contains basic runtime information on the algorithms. It shows how many enrichment suggestions were made per axiom type, the runtime of the algorithm, the average score and the average of the maximum scores of each algorithm run. The algorithms require 10-161 seconds per ontology entity. To the best of our knowledge, there are no other approaches performing the same task to which we can compare our method in general. For a small number of the reported axiom types [49] performs a similar approach using association rule mining, which yields very similar results to our approach due to high similarity of the underlying heuristics

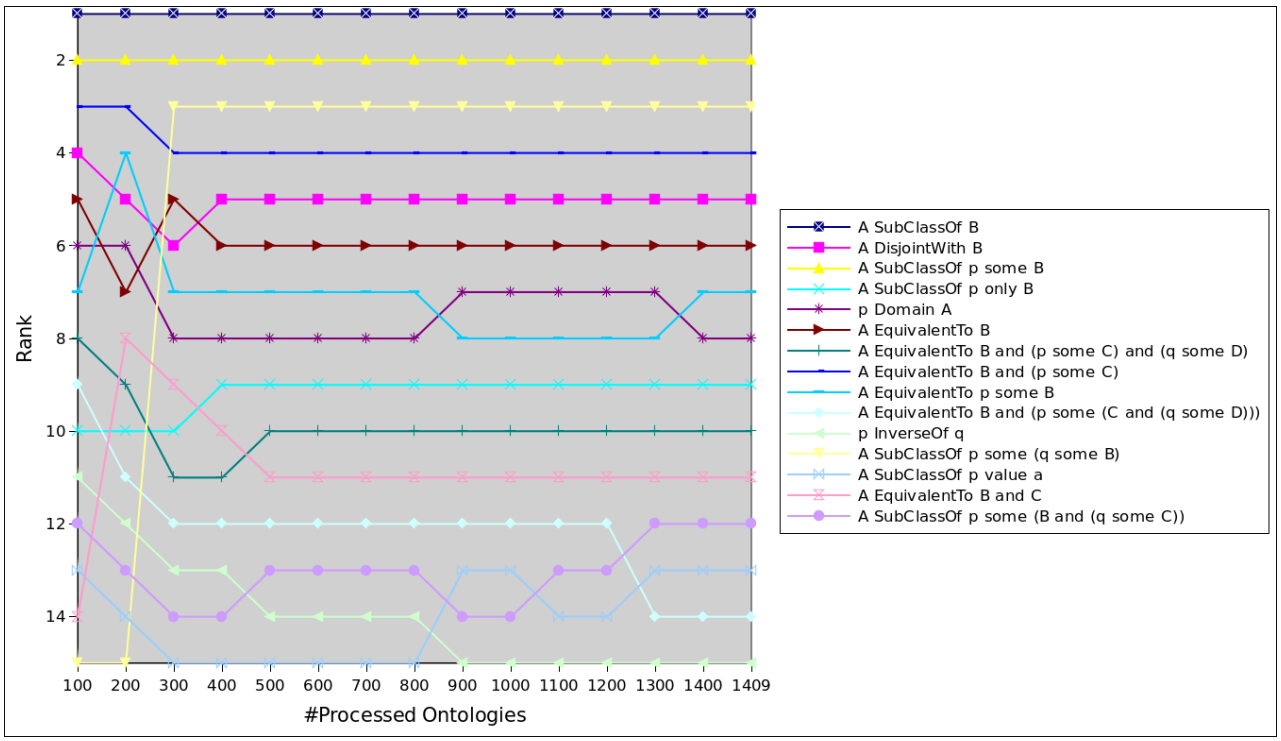


Figure 4: Top 15 axiom patterns and its sequence of rank when processing the ontologies in random order.

for this axiom types.

Table 6 shows our evaluation results. In this evaluation we defined recall with respect to the existing DBpedia ontology. For instance, $180/185$ in the subClassOf row indicates that we were able to re-learn 180 out of 185 such subclass relationships from the original DBpedia ontology. Higher numbers are an indicator that the methods do not miss many possible enrichment suggestions. The next column shows how many additional axioms were suggested, i.e. how many axioms were suggested which are not in the original DBpedia ontology. The last three columns are the result of a manual evaluation. Both authors independently observed at most 100 axioms per type (possibly less, in case fewer than 100 suggestions were made) and evaluated them manually. Three different categories were used: "yes" indicates that it is likely that they would be accepted by a knowledge engineer, "maybe" are corner cases and "no" are those, which would probably be rejected.

In summary, we observed that axioms regarding the class hierarchy basically seem to be more easy to learn than axioms building the property hierarchy. We also noticed that we could suggest new axioms for all axiom types except for the ReflexiveObjectProperty ones. The reason is that DBpedia does not contain corresponding instance data. general, we believe that reflexivity only amounts to a small proportion in most real world knowledge bases. The low recall for the range axioms of object and data properties is mostly due to either missing or different type information on the triples' objects.

9.3.2 Axiom Patterns

Table 7 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 7. 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

algorithm	Avg. nr. of #suggestions	Avg. runtime in ms	timeout in %	Avg. score	Avg. max. score
disjoint objectproperty	10.0	12384.0	0.16	1.00	1.00
equivalent objectproperty	1.1	12509.0	0.16	0.96	0.96
functional objectproperty	1.0	19990.0	0.48	0.89	0.89
inv.functional objectproperty	1.0	113590.0	4.29	0.86	0.86
objectproperty domain	3.1	11577.0	0.00	0.93	0.96
objectproperty range	2.6	14253.0	0.16	0.84	0.87
objectproperty subPropertyOf	1.1	56363.0	0.32	0.93	0.93
symmetric objectproperty	1.0	12730.0	0.32	0.80	0.80
transitive objectproperty	1.0	16830.0	1.91	0.84	0.84
irreflexive objectproperty	1.0	17357.0	0.16	0.97	0.97
reflexive objectproperty	0.0	10013.0	0.16	-	-
disjoint dataproperty	10.0	137204.0	3.97	1.00	1.00
equivalent dataproperty	1.0	161229.0	4.25	0.92	0.92
functional dataproperty	1.0	18281.0	0.42	0.94	0.94
dataproperty domain	2.8	10340.0	0.28	0.94	0.96
dataproperty range	1.0	13516.0	0.42	0.96	0.96
dataproperty subPropertyOf	1.0	91284.0	4.11	0.88	0.88
OVERALL	2.3	42909.9	1.31	0.92	0.92

Table 5: Basic information on runtime and number of suggestions of the algorithms.

and judged the axioms. An advantage of using DBpedia in this context is that 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.

			Estimated precision		
axiom type	recall	additional axioms	no	maybe	yes
SubObjectPropertyOf	0/0	45	18	9	18
EquivalentObjectProperties	0/0	40	40	0	0
DisjointObjectProperties	0/0	5670	0	0	100
ObjectPropertyDomain	385/449	675	10	22	68
ObjectPropertyRange	173/435	427	4	59	37
TransitiveObjectProperty	0/0	12	5	5	2
FunctionalObjectProperty	0/0	352	8	18	74
InverseFunctionalObjectProperty	0/0	173	72	3	25
SymmetricObjectProperty	0/0	3	0	0	3
ReflexiveObjectProperty	0/0	0	-	-	-
IrreflexiveObjectProperty	0/0	536	1	0	99
SubDataPropertyOf	0/0	197	86	8	6
EquivalentDataProperties	0/0	213	20	9	71
DisjointDataProperties	0/0	62	0	0	100
DataPropertyDomain	448/493	623	27	33	40
DataPropertyRange	118/597	79	0	0	100
FunctionalDataProperty	14/14	509	4	17	79

Table 6: Results of the manual evaluation for each property axiom type.

		manual evaluation in %			
pattern	sample size	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 7: Result of the manual evaluation for each axiom pattern.

9.4 Threshold Analysis

The diagram in Figure 5 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).

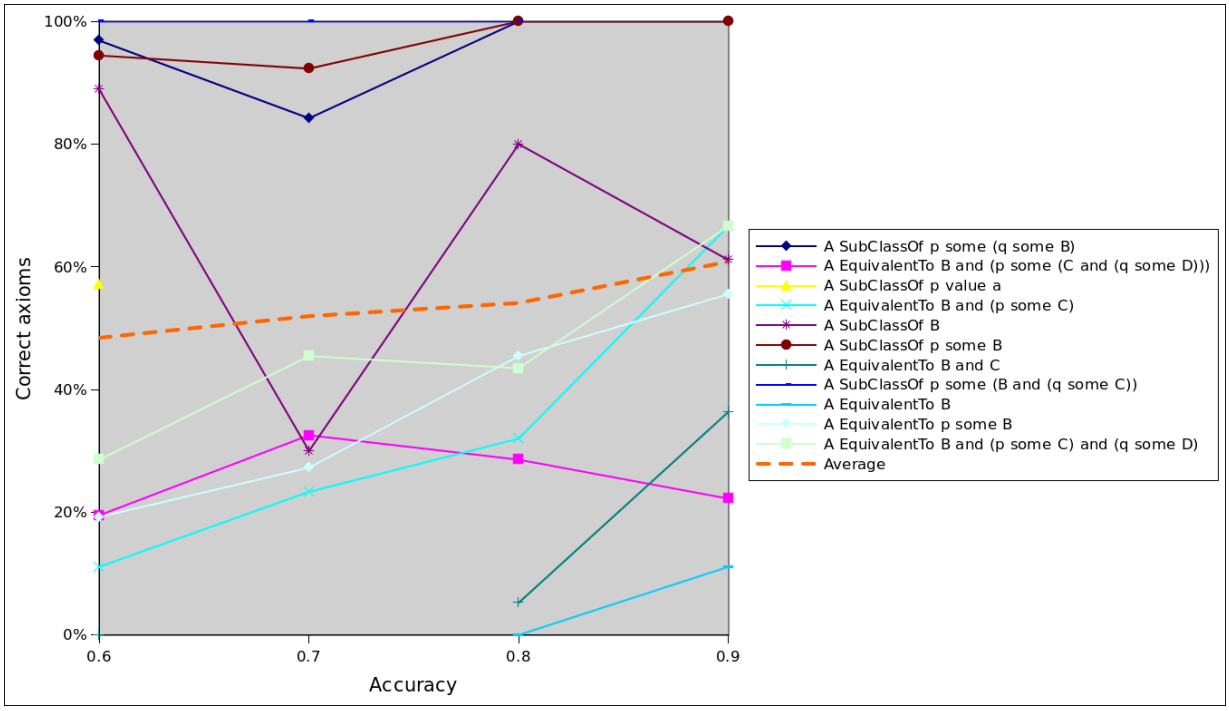


Figure 5: Correlation between the accuracy value of the pattern instantiations and the confidence of the evaluators.

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.

9.5 Discussion

In this part, we will explain some of the evaluation results and present specific examples.

9.5.1 Property Axioms

Below, we list some of the observations we made according to the property axioms evaluation described in Section 9.3.1:

- The test set for irreflexive properties contained `dbo:isPartOf`, which is usually considered as a reflexive property. It is the only incorrect suggestion in this test set.

- As an example for the imperfect recall on object property domains, the results of the learning procedure for `dbo:hometown` are as follows: For the existing domain `dbo:Person` we only got a score of 0.3, whereas `dbo:Band` and `dbo:Organisation` achieved a score of approx. 0.7. This is because each `dbo:Band` was also a `dbo:Person` at this time in DBpedia Live.
- We discovered 3 symmetric object properties, namely `dbo:neighboringMunicipality`, `dbo:sisterCollege` and `dbo:currentPartner`.
- For most of the data properties, the learned axioms of the types SubDataPropertyOf and EquivalentDataProperties contained properties of the DBpedia namespace `http://dbpedia.org/property/`(dbp), e.g. EquivalentDataProperties(`dbo:drugbank`,`dbp:drugbank`). Mixing the two different ontologies is usually not desirable, hence the low precision in some of these cases. As a result, we now support more fine-grained control over the used ontology namespaces to avoid those problems.
- We missed some DataPropertyRanges, because sometimes the defined range in the ontology is a different datatype, compared to the one of the literal values in the triples. For instance `dbo:background` has a defined range `xsd:string`, but in the instance data the literals only have a language tag (which makes them implicit to `rdf:PlainLiteral`). `dbo:budget` (range in ontology: `xsd:double`, but `http://dbpedia.org/datatype/usDollar` used in the actual literals) is a different example. Clearly, in those cases data errors cause problems and our enrichment tool can be used to detect those.
- In some cases we learned a different datatype, so we missed the existing one and found an additional one. Most of these additional axioms would also be a reasonable but not optimal choice, e.g. for `dbo:populationTotal` we learned `xsd:integer`, whereas in the ontology `xsd:nonNegativeInteger` is defined.

9.5.2 Axiom Patterns

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 $\texttt{GrandPrix} \equiv \texttt{Event} \sqcap (\exists \texttt{poleDriver.Athlete}) \sqcap (\exists \texttt{secondDriver.Agent})$. While logically correct, defining a grand prix via the `Agent` class relation ship of the driver finishing second is not intuitive.

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

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

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.

10 Related Work

10.1 Ontology Enrichment

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) [42] and more specifically supervised learning in description logics. sciences to detect whether drugs are likely to be efficient for particular diseases. Work on learning in description logics goes back to e.g. [13, 14], which used so-called *least common subsumers* to solve the learning problem (a modified variant of the problem defined in this article). Later, [6] invented a refinement operator for $\mathcal{ALER}$ and proposed to solve the problem by using a top-down approach. Early techniques [13] using *least common subsumers* were later enriched with refinement operators [15, 23, 24] 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 [37] 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 [16] is a similar approach, which is based on a mixture of upward and downward refinement of class expressions. take the open world assumption into account, which was not done in ILP previously. Most recently, [30] implements appropriate heuristics and adaptations for learning definitions in ontologies. which improve previous methods by an order of magnitude. For this reason, we will analyse it in more detail in the next subsection. The algorithms presented in the article can also learn *super class axioms*.

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) [5] 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. [38] focus on learning in hybrid knowledge bases combining ontologies and *rules*. rules at the cost of efficiency (because of the larger search space). Similar as in knowledge representation, the tradeoff between expressiveness of the target language and efficiency of learning algorithms is a critical choice in symbolic machine learning.

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 [44] and further pursued in e.g. [4] investigates the use of *formal concept analysis* for completing knowledge bases. [50] proposes to improve knowledge bases through relational exploration and implemented it in the *RELExO framework*[16]. 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.

[51] 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.

survey on ontology mapping.

[16] http://code.google.com/p/relexo/

Type/Aim	References
Taxonomies	[52, 8, 49]
Definitions	often done via ILP approaches e.g. [33, 10, 25, 36, 35, 37, 30, 28, 29, 34, 27, 16, 6], genetic approaches [27] have also been used, optimisation and applications can be found in [11, 22, 21, 20, 32]
Super Class Axioms	[30, 49, 8]
Rules in Ontologies	[38, 39]
Disjointness	[51, 8, 47]
Properties of Properties	[8, 17, 47]
Alignment	challenges: [46], recent survey: [12]
Completion	formal concept analysis and relational exploration [4, 50, 45]

Table 8: Work in ontology enrichment grouped by type or aim of learned structures.

There are further more light-weight ontology enrichment methods. For instance, *taxonomies* can be learned from simple tag structures via heuristics [8, 49, 47]. All of those approaches follow similar goals. [8] is the base of this article and follows the approach described in Section 2. [49] 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, [47] 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.

10.2 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 [19] for a systematic review on ontology design patterns articles in the Semantic Web community and [18] 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 [7]. 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.

11 Conclusions and Future Work

We presented an approach, which allows to detect property axioms and frequent axiom usage patterns using ≈ 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. Overall, 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.

Pointers

DL-Learner: http://dl-learner.org

ORE: http://ore-tool.net

References

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

[2] Sören Auer, Chris Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a web of open data. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2008.

[3] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] Franz Baader, Bernhard Ganter, Ulrike Sattler, and Baris Sertkaya. Completing description logic knowledge bases using formal concept analysis. In *IJCAI 2007*. AAAI Press, 2007.

[5] Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. Applied Logic*, 5(3):392–420, 2007.

[6] Liviu Badea and Shan-Hwei Nienhuys-Cheng. A refinement operator for description logics. In *ILP 2000*, volume 1866 of *LNAI*, pages 40–59. Springer, 2000.

[7] Eva Blomqvist. Ontocase-automatic ontology enrichment based on ontology design patterns. In *The Semantic Web-ISWC 2009*, pages 65–80. Springer, 2009.

[8] Lorenz Bühmann and Jens Lehmann. Universal OWL axiom enrichment for large knowledge bases. In *Proceedings of EKAW 2012*, pages 57–71. Springer, 2012.

[9] Lorenz Bühmann and Jens Lehmann. OWL class expression to SPARQL rewriting. Technical report, University of Leipzig, 2013. http://svn.aksw.org/papers/2013/OWL_SPARQL/public.pdf.

[10] Lorenz Buhmann and Jens Lehmann. Pattern based knowledge base enrichment. In *12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia*, 2013.

[11] Didier Cherix, Sebastian Hellmann, and Jens Lehmann. Improving the performance of a sparql component for semantic web applications. In *JIST*, 2012.

[12] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, 2006.

[13] William W. Cohen, Alexander Borgida, and Haym Hirsh. Computing least common subsumers in description logics. In *AAAI 1992*, pages 754–760, 1992.

[14] William W. Cohen and Haym Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In *KR 1994*, pages 121–133. Morgan Kaufmann, 1994.

[15] Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. In *ISWC 2004*, pages 441–455. Springer, 2004.

[16] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL concept learning in description logics. In *ILP 2008*, volume 5194 of *LNCS*, pages 107–121. Springer, 2008.

[17] Daniel Fleischhacker, Johanna Völker, and Heiner Stuckenschmidt. Mining rdf data for property axioms. In *OTM Conferences (2)*, pages 718–735, 2012.

[18] Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on Ontologies*, pages 221–243. Springer, 2009.

[19] Karl Hammar and Kurt 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.

[20] Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL class descriptions on very large knowledge bases. *International Journal on Semantic Web and Information Systems*, 5(2):25–48, 2009.

[21] Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of owl class expressions on very large knowledge bases and its applications. In Interoperability Semantic Services and Web Applications: Emerging Concepts, editors, *Learning of OWL Class Expressions on Very Large Knowledge Bases and its Applications*, chapter 5, pages 104–130. IGI Global, 2011.

[22] Sebastian Hellmann, Jens Lehmann, Jörg Unbehauen, Claus Stadler, Thanh Nghia Lam, and Markus Strohmaier. Navigation-induced knowledge engineering by example. In *JIST*, 2012.

[23] Luigi Iannone and Ignazio Palmisano. An algorithm based on counterfactuals for concept learning in the semantic web. In *IEA/AIE 2005*, pages 370–379, June 2005.

[24] Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.

[25] Josué Iglesias and Jens Lehmann. Towards integrating fuzzy logic capabilities into an ontology-based inductive logic programming framework. In *Proc. of the 11th International Conference on Intelligent Systems Design and Applications (ISDA)*, 2011.

[26] Yong-Bin Kang, Yuan-Fang Li, and Shonali 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.

[27] Jens 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.

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

[29] Jens Lehmann. *Learning OWL Class Expressions*, volume 6 of *Studies on the Semantic Web*. AKA Heidelberg, 2010. ISBN 978-3-89838-336-3.2010.

[30] Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9:71 – 81, 2011.

[31] Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[32] Jens Lehmann and Lorenz 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.

[33] Jens Lehmann and Lorenz Bühmann. Autosparql: Let users query your knowledge base. In *Proceedings of ESWC 2011*, 2011.

[34] Jens Lehmann and Christoph Haase. Ideal downward refinement in the EL description logic. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*, 2009.

[35] Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In Hendrik Blockeel, Jan Ramon, Jude W. Shavlik, and Prasad 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.

[36] Jens Lehmann and Pascal 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.

[37] Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250, 2010.

[38] Francesca 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.

[39] Francesca A. Lisi and Floriana Esposito. Learning SHIQ+log rules for ontology evolution. In *SWAP 2008*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[40] Eleni Mikroyannidi, Nor Azlinayati Abdul Manaf, Luigi Iannone, and Robert Stevens. Analysing syntactic regularities in ontologies. In Pavel Klinov and Matthew Horridge, editors, *OWLED*, volume 849 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[41] Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, and Sebastian Hellmann. DBpedia and the Live Extraction of Structured Data from Wikipedia. *Program: electronic library and information systems*, 46:27, 2012.

[42] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *LNCS*. Springer, 1997.

[43] Daniel L. Rubin, Dilvan A. Moreira, Pradip Kanjamala, and Mark 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.

[44] Sebastian Rudolph. Exploring relational structures via FLE. In Karl Erich Wolff, Heather D. Pfeiffer, and Harry S. Delugach, editors, *ICCS 2004*, volume 3127 of *LNCS*, pages 196–212. Springer, 2004.

[45] Baris Sertkaya. OntocomP system description. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike 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.

[46] Pavel Shvaiko and Jerome Euzenat. Ten challenges for ontology matching. Technical report, August 01 2008.

[47] Gerald Töpper, Magnus Knuth, and Harald Sack. Dbpedia ontology enrichment for inconsistency detection. In *Proceedings of the 8th International Conference on Semantic Systems*, pages 33–40. ACM, 2012.

[48] Chiara Del Vescovo, Damian Gessler, Pavel Klinov, Bijan Parsia, Ulrike Sattler, Thomas Schneider 0002, and Andrew Winget. Decomposition and modular structure of bioportal ontologies. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2011.

[49] Johanna Völker and Mathias Niepert. Statistical schema induction. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff 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.

[50] Johanna Völker and Sebastian Rudolph. Fostering web intelligence by semi-automatic OWL ontology refinement. In *Web Intelligence*, pages 454–460. IEEE, 2008.

[51] Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning disjointness. In *ESWC 2007*, volume 4519 of *LNCS*, pages 175–189. Springer, 2007.

[52] Harris Wu, Mohammad Zubair, and Kurt 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.